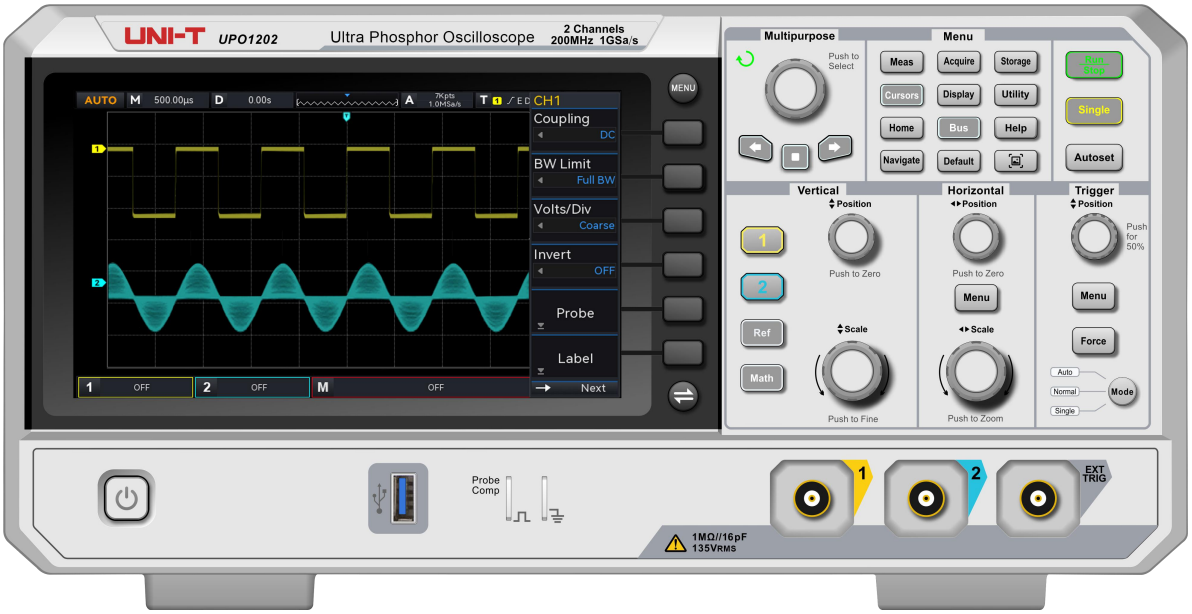
Instruments.uni-trend.com



Programming Manual

UPO1002 Series Digital Phosphor Oscilloscope

REV 0

2023.11.2

Warranty and Statement

Copyright

Brand Information

UNI-T is the registered trademark of Uni-Trend Technology (China) Co., Ltd.

File Number

20231120

Software Version

V1.00.0003

Software upgrade may have some change and add more function, please subscribe UNI-T website to get the newest manual or contact UNI-T to update the version.

Statement

- UNI-T products are protected by patents (including obtained and pending) in China and other countries and regions.
- UNI-T reserves the right to change specifications and prices.
- The information provided in this manual supersedes all previous publications.
- Information provided in this manual is subject to change without prior notice.
- **UNI-T** shall not be liable for any errors that may be contained in this manual. For any incidental or consequential damages, arising out of the use or the information and deductive functions provided in this manual.
- Without the written permission of **UNI-T**, this manual cannot photocopied, reproduced or adapted.

Product Certification

UNI-T has certified that the product conforms to China national product standard and industry product standard as well as ISO9001:2008 standard and ISO14001:2004 standard. **UNI-T** will go further to certificate product to meet the standard of other member of the international standards

organization.

Contact Us

If you have any question or problem, please contact **UNI-T**.

Mail: infosh@uni-trend.com.cn

Official Website:https://www.uni-trend.com

This programming manual is suitable for UPO1002 series.

SCPI

SCPI (Standard Commands for Programmable Instruments) is a standardized instrument programming language that builds on existing standards IEEE 488.1 and IEEE 488.2 and follows the floating point rules of IEEE 754 standard, ISO 646 message exchange 7-bit encoding notation (equivalent to ASCII programming) and many other standards.

This section introduces the format, symbols, parameters, and abbreviations of the SCPI command.

Instruction Format

The SCPI command is a tree-like hierarchy consisting of multiple subsystems, each subsystem consisting of a root keyword and one or more hierarchical key words. **The command line usually begins with a colon ":"; Keywords are separated by the colon ":", followed by optional parameter settings. The command keyword is separated by spaces from the first parameter. The command string must end with a newline <NL> character. Add the question mark "? " after the command line. It is usually indicated that this feature is being queried.**

Symbol Description

The following four symbols are not part of the SCPI command. They cannot be sent with the command, but they are commonly used for supplementary specification.

- **Braces { }**

  It usually contains multiple optional parameter, one of which must be selected when send a command.

  For example, :DISPlay:GRID:MODE { FULL | GRID | CROSS | NONE}

- **Vertical Bar |**

  It is used to separate multiple parameters, one of which must be selected when send a command.

  For example, :DISPlay:GRID:MODE { FULL | GRID | CROSS | NONE}

- **Square Brackets [ ]**

  The parameter in square brackets (command keywords) can be omitted. If the parameter is omitted, the instrument will set the parameter to the default value.

For example, the command :MEASure:NDUTy? [<source>], [<source>] represents the current channel.

- **Angle Braces < >**

  The parameter in angle braces must be replaced by a valid value.

  For example, the command DISPlay:GRID:BRIGhtness <count> is sent as the format of DISPlay:GRID:BRIGhtness 30.

Parameter Description

The parameter in this manual can be divided into five types: Boolean, Integer, Real, Discrete and ASCII string.

- **Boolean**

  The parameter takes value as "ON" (1) or "OFF" (0).

  For example, :SYSTem:LOCK {{1 | ON} | {0 | OFF}}

- **Integer**

  Unless other specified, the parameter can take any integer that within the valid range. Note: Do not set the parameter as decimal format, otherwise, an error will occur.

  For example, the parameter < count > in the command of :DISPlay:GRID:BRIGhtness <count> can be any integer value within the range of 0-100.

- **Real**

  Unless other specified, the parameter can take any integer that within the valid range.

  For example, for CH1, the parameter <offset> in the command of CHANnel1:OFFSet <offset> is real type.

- **Discrete**

  The parameter can only be specified value or character.

  For example, the parameter in the command :DISPlay:GRID:MODE { FULL | GRID | CROSS | NONE} can only be FULL, GRID, CROSS, NONE.

- **ASCIIString**

  The character string can virtually contain all ASCII character sets. The character string must starts and ends with paired quotation marks; single or double quotes can be used. The separator character can also be used as a part of character string by typing it twice without adding any characters.

For example, set IP: SYST:COMM:LAN:IPAD “192.168.1.10”

Abbreviation

All the commands are case-insensitive. The commands can be all input in uppercase letters or in lowercase letters. For abbreviations, it should enter all the uppercase letters that exist in the command syntax.

Data Return

Data return is divided into single data and batch data. The single data returns the corresponding parameter type, in which the real return type is represents by the scientific notation method. The part before e retains three figure behind the decimal point, and the e part retains three figure; the batch data return must be obey IEEE 488.2# string data format, ‘#’+ the length of character bits [fixed to one character] + ASCII valid value+ valid data+ end mark [‘\n’].

For example, #3123xxxxxxxxxxxxxxxxxxx\n represents the return format of valid batch data with 123 bytes, here '3' means that the '123' occupies 3 character positions.

Note: If returned data is invalid, use * to represent it.

SCPI Explanation

IEEE488.2 Common Command

*IDN?

- **Command format**

  *IDN?

- **Functional description**

  Query the manufacturer name, oscilloscope's model, product serial number and software version.

- **Return format**

  The query returns the manufacturer name, oscilloscope's model, product serial number, which separated by dot mark.

- **For example**

  UNI-T Technologies, UPO1000X, UPO1000, 00.00.01

*RST

- **Command format**

  *RST

- **Functional description**

  Restore the instrument to its factory settings, clear all the error message and send and receive queue buffers.

*OPC

- **Command format**

  *OPC

  *OPC?

- **Functional description**

  Force the current command that has been executed to mark at position 1.

- **Return format**

  Query returns whether the currently sent command is executed. 1 indicates completed, 0 indicates not completed.

- **For example**

  *OPC Mark the completed command at position 1

  *OPC? Query returns 1, which indicates that the current command is executed, otherwise, it's

not.

SYSTem Command

The command is used for the basic operation of oscilloscope, including operating control, full keyboard lock, error queue and system data setting.

:RUN

- **Command format**

  :RUN

- **Functional description**

  Start the sampling operating of oscilloscope. Executing the command :STOP can stop the operation.

:STOP

- **Command format**

  :STOP

- **Functional description**

  Stop the sampling operating of oscilloscope. Executing the command :RUN can resume the operation.

:AUTO

- **Command format**

  :AUTO

- **Functional description**

  Automatically set the control parameter of the instrument, the input waveform can achieve the best display effect through automatic setting

:SYSTem:LOCK

- **Command format**

  :SYSTem:LOCK {{1 | ON} | {0 | OFF}}

  :SYSTem:LOCK?

- **Functional description**

  Lock/unlock full keyboard.

- **Return format**

  Query returns the lock state of full keyboard, 0 represents unlock, 1 represents locked.

- **For example**

:SYSTem:LOCK ON/:SYST:LOCK 1 Full keyboard is locked

:SYSTem:LOCK OFF/:SYST:LOCK 0 Unlock full keyboard

:SYSTem:LOCK?

Query returns 1, which indicates full keyboard is locked

:SYSTem:ERRor

- **Command format**

:SYSTem:ERRor

:SYSTem:ERRor?

- **Functional description**

Clear error message queue.

- **Return format**

Query returns the last message error with the format of "<Message number>, <Message content>". <Message number> is an integer, <Message content> is a ASCII string with double quotation marks.

Such as, -113, which indicating that "Undefined header; command cannot be found."

- **For example**

:SYSTem:ERR	Clear error message queue
:SYSTem:ERR?	Query returns: -113, which indicating that "Undefined header; command cannot be found."

:SYSTem:SETup

- **Command format**

:SYSTem:SETup <setup_data>

:SYSTem:SETup?

- **Functional description**

Configure the system setup data, <setup_data> is comform with Appendix 2:IEEE 488.2 Binary Data Format.

- **Return format**

Query returns the system setup data.

:SYSTem:LANGuage

- **Command format**

:SYSTem:LANGuage { ENGLish | SIMPlifiedchinese| GERMan | FRENch | POLish | SPANish | ITALian}

:SYSTem:LANGuage?

- **Functional description**

The system language can set English, simplified Chinese, German, France, Polish, Spanish, Italian.

- **Return format**

Query returns { ENGLish | SIMPlifiedchinese| GERMan | FRENch | POLish | SPANish | ITALian}.

- **For example**

:SYSTem:LANGuage ENGL Set the system language to ENGL

:SYSTem:LANGuage? Query returns ENGLish

:SYSTem:RTC

- **Command format**

:SYSTem:RTC <year>,<month>,<day>,<hour>,<minute>,<second>

:SYSTem:RTC?

- **Functional description**

Set the system time.

- **Return format**

Query returns year, month, day, hour, minute, second.

- **For example**

:SYSTem:RTC 2017,7,7,20,8,8 Set the system time as 20:08:08, 7th, July, 2017

:SYSTem:RTC? Query returns 2017,7,7,20,8,8

:SYSTem:CAL

- **Command format**

:SYSTem:CAL

- **Functional description**

Set system self-calibration. During the self-calibration, the communication cannot be normal used.

:SYSTem:CLEAr

- **Command format**

:SYSTem:CLEAr

- **Functional description**

Clear all saved storage waveform and setup data of system.

:SYSTem:CYMOmeter

- **Command format**

  :SYSTem:CYMOmeter {1 | ON} | {0 | OFF}

  :SYSTem:CYMOmeter?

- **Functional description**

  Turn on/off frequency meter.

- **Return format**

  Query returns the state of frequency meter, 1 represents ON, 0 represents OFF.

- **For example**

  :SYSTem:CYMOmeter ON Turn on frequency meter

  :SYSTem:CYMOmeter? Query returns 1

:SYSTem:CYMOmeter:FREQuency?

- **Command format**

  :SYSTem:CYMOmeter:FREQuency?

- **Functional description**

  Acquiring the frequency value of frequency meter, it can be obtained without opening the frequency meter.

- **Return format**

  Query returns the frequency value of frequency meter, it returns * when no valid value.

- **For example**

  :SYSTem:CYMOmeter:FREQuency? Query returns 1.20000E+3

:SYSTem:SQUare:SELect

- **Command format**

  :SYSTem:SQUare:SELect { 10Hz | 100Hz | 1KHz | 10KHz }

  :SYSTem:SQUare:SELect?

- **Functional description**

  Select square wave output.

- **Return format**

  Query returns { 10Hz | 100Hz | 1KHz | 10KHz }.

- **For example**

  :SYSTem:SQUare:SELect 10Hz Select square wave output of 10Hz

:SYSTem:SQUare:SELect? Query returns 10Hz

:SYSTem:OUTPut:SELect

- **Command format**

  :SYSTem:OUTPut:SELect { TRIGger | PASS_FAIL | DVM }

  :SYSTem:OUTPut:SELect?

- **Functional description**

  Select output to be TRIGger, PASS_FAIL or DVM (digital voltmeter).

- **Return format**

  Query returns { TRIGger | PASS_FAIL | DVM }.

- **For example**

  :SYSTem:OUTPut:SELect TRIGger Select the output to be TRIGger

  :SYSTem:OUTPut:SELect? Query returns TRIGger

:SYSTem:BOOT:LOAD

- **Command format**

  :SYSTem:BOOT:LOAD {DEFault | LAST }

  :SYSTem:BOOT:LOAD?

- **Functional description**

  Set loading mode for power-on, DEFault represents default setting, LAST represents the last setting.

- **Return format**

  Query returns {DEFault | LAST }.

- **For example**

  :SYSTem:BOOT:LOAD LAST Loading the last setting when the instrument is power-on

  :SYSTem:BOOT:LOAD? Query returns LAST

:SYSTem:MNUDisplay

- **Command format**

  :SYSTem:MNUDisplay { 5S | 10S | 20S | INFinite}

  :SYSTem:MNUDisplay?

- **Functional description**

  Set the display time of menu, INFinite represents the menu will display all the time.

- **Return format**

  Query returns { 5S | 10S | 20S | INFinite }.

- **For example**

:SYSTem:MNUDisplay 5S Set the menu display time to 5s, the menu will automatically withdraw after 5s

:SYSTem:MNUDisplay? Query returns 5s

:SYSTem:BRIGhtness

- **Command format**

:SYSTem:BRIGhtness <count>

:SYSTem:BRIGhtness?

- **Functional description**

Set the screen brightness, <count> takes value as 10~100, the bigger the number, the brighter the screen.

- **Return format**

Query returns the current screen brightness.

- **For example**

:SYSTem:BRIGhtness 50 Set the screen brightness to 50

:SYSTem:BRIGhtness? Query returns 50

:SYSTem:VERSion?

- **Command format**

:SYSTem:VERSion?

- **Return format**

Query returns the version information, which are 128 bytes character string.

HW is hardware version number, SW is software version number, PD is created date, ICV is protocol version number.

- **For example**

:SYST:VERS? Query returns

HW:1.0;SW:1.0;PD:2014-11-20;ICV:1.4.0

:SYSTem:COMMunicate:LAN:APPLy

- **Command format**

:SYSTem:COMMunicate:LAN:APPLy

- **Functional description**

The current network parameter will take effect immediately.

:SYSTem:COMMunicate:LAN:GATEway

- **Command format**

  :SYSTem:COMMunicate:LAN:GATEway <gateway>

  :SYSTem:COMMunicate:LAN:GATEway?

- **Functional description**

  Set the default gateway, <gateway> is belong to the ASCII character string, the format is xxx.xxx.xxx.xxx.

- **Return format**

  Query returns the default gateway.

- **For example**

  :SYST:COMM:LAN:GATE "192.168.1.1" Set the default gateway 192.168.1.1

  :SYST:COMM:LAN:GATE? Query returns 192.168.1.1

:SYSTem:COMMunicate:LAN:SMASK

- **Command format**

  :SYSTem:COMMunicate:LAN:SMASK <submask>

  :SYSTem:COMMunicate:LAN:SMASK?

- **Functional description**

  Set the subnet mask, <submask> is belong to the ASCII character string, the format is xxx.xxx.xxx.xxx.

- **Return format**

  Query returns the subnet mask.

- **For example**

  :SYST:COMM:LAN:SMASK "255.255.255.0"

  Set the subnet mask 255.255.255.0

  :SYST:COMM:LAN:SMASK? Query returns 255.255.255.0

:SYSTem:COMMunicate:LAN:IPADdress

- **Command format**

  :SYSTem:COMMunicate:LAN:IPADdress <ip>

  :SYSTem:COMMunicate:LAN:IPADdress?

- **Functional description**

  Set IP address, <ip> is belong to the ASCII character string, the format is xxx.xxx.xxx.xxx.

- **Return format**

  Query returns IP address.

- **For example**

  :SYST:COMM:LAN:IPAD "192.168.1.10" Set IP address 192.168.1.10

  :SYST:COMM:LAN:IPAD? Query returns 192.168.1.10

:SYSTem:COMMunicate:LAN:DHCP

- **Command format**

  :SYSTem:COMMunicate:LAN:DHCP {{1 | ON} | {0 | OFF}}

  :SYSTem:COMMunicate:LAN:DHCP?

- **Functional description**

  Switch the configuration mode to automatic IP or manual IP.

- **Return format**

  Query returns dynamic configuration mode, 0 represents manual IP，1 represents automatic IP.

- **For example**

  :SYST:COMM:LAN:DHCP ON Turn on IP dynamic configuration mode

  :SYST:COMM:LAN:DHCP? Query returns 1

:SYSTem:COMMunicate:LAN:MAC?

- **Command format**

  :SYSTem:COMMunicate:LAN:MAC?

- **Return format**

  Query returns MAC physical address.

- **For example**

  :SYST:COMM:LAN:MAC? Query returns 00-2A-A0-AA-E0-56

:SYSTem:AUTO:ACQuire

- **Command format**

  :SYSTem:AUTO:ACQuire {AUTO|KEEP}

  :SYSTem:AUTO:ACQuire?

- **Functional description**

  Set whether the sampling mode keeps the current status in automatic setting.

  AUTO represents that the sampling mode is automatically set by the preset parameter; KEEP represents that the sampling mode is automatically set based on the current status.

- **Return format**

  Query returns {AUTO|KEEP}.

- **For example**

:SYSTem:AUTO:ACQuire KEEP

The sampling mode keeps the current automatic setting

:SYSTem:AUTO:ACQuire? Query returns KEEP

:SYSTem:AUTO:TRIGger

- **Command format**

:SYSTem:AUTO:TRIGger {AUTO|KEEP}

:SYSTem:AUTO:TRIGger?

- **Functional description**

Set whether the trigger coupling keeps the current status in automatic setting.

AUTO represents that the trigger is automatically set by the preset parameter; KEEP represents that the trigger coupling is automatically set based on the current status.

- **Return format**

Query returns {AUTO|KEEP}.

- **For example**

:SYSTem:AUTO:TRIGger KEEP

The trigger keeps the current automatic setting

:SYSTem:AUTO:TRIGger? Query returns KEEP

:SYSTem:AUTO:SIGNal

- **Command format**

:SYSTem:AUTO:SIGNal {AUTO|KEEP}

:SYSTem:AUTO:SIGNal?

- **Functional description**

Set whether the input channel (active channel) keeps the current status in automatic setting.

AUTO represents that the active channel is automatically set by the preset parameter; KEEP represents that active channel is automatically set based on the current status.

- **Return format**

Query {AUTO|KEEP}.

- **For example**

:SYSTem:AUTO:SIGNal KEEP

The active channel keeps the current automatic setting

:SYSTem:AUTO:SIGNal? Query returns KEEP

KEY Command

This command is used to control the key and rotary knob on the operating panel of the oscilloscope.

:KEY:<key>

- **Command format**

  :KEY:<key>

  :KEY:<key>:LOCK { {1 | ON} | {0 | OFF} }

  :KEY:<key>:LOCK?

  :KEY:<key>:LED?

- **Functional description**

  Set key function and lock/unlock key function. <key> definition and description refer to Appendix 1:Key List.

- **Return format**

  Query returns the status of lock key or the status of LED key.

  Lock status: 0 represents the key is unlock, 1 represents the key is locked

  LED status: 0 represents LED indicator is extinguished, 1 represents LED indicator is illumined.

- **For example**

  :KEY:CH1 CH1 key is pressed

  :KEY:CH1:LOCK ON/OFF Lock/unlock key

  :KEY:CH1:LOCK?

  Query returns the lock status of key, 1 represents the key is locked

  :KEY:CH1:LED?

  Query returns the status of LED, 0 represents LED indicator is extinguished

CHANnel Command

This command is used to separately set each channel.

:CHANnel<n>:BWLimit

- **Command format**

  :CHANnel<n>:BWLimit {20M | FULL}

  :CHANnel<n>:BWLimit?

- **Functional description**

  Set the bandwidth limit function.

20M: turn on the bandwidth limit to 20 MHz, to reduce display noise.

FULL: turn on the bandwidth limit to full display.

<n>: {1|2}, it represents {CH1|CH2} respectively.

- **Return format**

Query returns {20M | FULL}.

- **For example**

:CHAN1:BWL 20M	Turn on 20 MHz (bandwidth limit) of CH1
:CHAN1:BWL?	Query returns 20 M

:CHANnel<n>:COUPling

- **Command format**

:CHANnel<n>:COUPling {DC|AC|GND}

:CHANnel<n>:COUPling?

- **Functional description**

Set the coupling mode for channel.

DC represents AC and DC component of the input signal.

AC represents the DC component that blocks the input signal.

GND represents cut-off the input signal.

<n>: {1|2}, it represents {CH1|CH2} respectively.

- **Return format**

Query AC, DC or GND.

- **For example**

:CHAN1:COUP DC	Set the coupling mode of CH1 to DC
:CHAN1:COUP?	Query returns DC

:CHANnel<n>:DISPlay

- **Command format**

:CHANnel<n>:DISPlay { {1|ON} | {0|OFF} }

:CHANnel<n>:DISPlay?

- **Functional description**

Turn on/off the specified channel.

<n>: {1|2}, it represents {CH1|CH2} respectively.

- **Return format**

Query returns 1 or 0, it represents ON or OFF respectively.

- **For example**

:CHAN1:DISP ON	Turn on CH1
:CHAN1:DISP?	Query returns 1, it represents that CH1 is opened

:CHANnel<n>:INVert

- **Command format**

  :CHANnel<n>:INVert { {1|ON} | {0|OFF} }

  :CHANnel<n>:INVert?

- **Functional description**

  Turn on/off reversed phase of waveform, ON (reversed phase is enabled), OFF (normal display).

  <n>: {1|2}, it represents {CH1|CH2} respectively.

- **Return format**

  Query returns 1 or 0, it represents ON or OFF respectively.

- **For example**

  :CHAN1:INV OFF Turn off reversed phase of CH1

  :CHAN1:INV?

  Query returns 0， it represents that reversed phase of CH1 is closed

:CHANnel<n>:PROBe:TYPe

- **Command format**

  :CHANnel<n>:PROBe:TYPe { VOLTs | AMPeres }

  :CHANnel<n>:PROBe:TYPe?

- **Functional description**

  Set the probe type, VOLTs represents voltage, AMPeres represents current.

  <n>: {1|2}, it represents {CH1|CH2} respectively.

- **Return format**

  Query returns the oscilloscope's type.

- **For example**

:CHAN1:PROB:TYP AMPeres	Set the probe of CH1 to be current probe
:CHAN1:PROB:TYP?	Query returns AMPeres

:CHANnel<n>:PROBe

- **Command format**

  :CHANnel<n>:PROBe { <probe> }

  :CHANnel<n>:PROBe?

- **Functional description**

Set the probe attenuation factor, the range is 0.001X~20000X.

<probe>: customized probe attenuation factor

<n>: {1|2}, it represents {CH1|CH2} respectively.

- **Return format**

Query returns the probe attenuation factor of oscilloscope by scientific notation and the unit is X.

- **For example**

:CHAN1:PROB 10X	Set probe attenuation factor of CH1 to 10X
:CHAN1:PROB?	Query returns 1.000000e+01

:CHANnel<n>:OFFSet

- **Command format**

:CHANnel<n>:OFFSet <offset>

:CHANnel<n>:OFFSet?

- **Functional description**

Set the waveform offset on vertical direction.

<n>: {1|2|5|6|7}, it represents { CH1|CH2|MATH|REFA|REFB} respectively.

- **Return format**

Query returns the set value of offset by scientific notation and the unit is V.

- **For example**

:CHAN1:OFFS 20V	Set the vertical offset of CH1 to 20 V
:CHAN1:OFFS?	Query returns 2.000e001

:CHANnel<n>:SCALe

- **Command format**

:CHANnel<n>:SCALe {<scale> | UP | DOWN}

:CHANnel<n>:SCALe?

- **Functional description**

Set the volts/div scale of oscilloscope on vertical direction.

<scale>: volts/div

UP: increase one scale based on the current scale

DOWN: decrease one scale based on the current scale

<n>: {1|2|5|6|7}, it represents { CH1|CH2|MATH|REFA|REFB} respectively.

- **Return format**

Query returns the current value of volts/div scale by scientific notation and the unit is V.

- **For example**

:CHAN1:SCAL 20V	Set volts/div scale of CH1 to 20 V
:CHAN1:SCAL?	Query returns 2.000e001
:CHAN1:SCAL UP	Increase one scale based on 20 V

:CHANnel<n>:UNITs

- **Command format**

:CHANnel<n>:UNITs {VOLTs|AMPeres|WATTs|UNKNown}

:CHANnel<n>:UNITs?

- **Functional description**

Set the channel's unit to VOLTs (voltage), AMPeres (current), WATTs (power) or UNKNown.

<n>: {1|2}, it represents {CH1|CH2} respectively.

- **Return format**

Query returns VOLTs, AMPeres, WATTs or UNKNown.

- **For example**

:CHAN1:UNIT VOLT	Set CH1's unit to VOLTs (voltage)
:CHAN1:UNIT?	Query returns VOLTs

:CHANnel<n>:VERNier

- **Command format**

:CHANnel<n>:VERNier { {1|ON} | {0|OFF} }

:CHANnel<n>:VERNier?

- **Functional description**

Set adjusting mode for the scale.

ON (Fine): fine tuning is used to adjusting the scale more exquisite for improving vertical resolution.

OFF (Coarse): set the vertical sensitivity by 1-2-5 system

<n>: {1|2}, it represents {CH1|CH2} respectively.

- **Return format**

Query returns 1 or 0, it represents ON or OFF respectively.

- **For example**

:CHAN1:VERN ON	Turn on fine tuning of CH1
:CHAN1:VERN?	Query returns 1

:CHANnel<n>:SELect

- **Command format**

  :CHANnel<n>:SELect

  :CHANnel<n>:SELect?

- **Functional description**

  Select a channel.

  <n>: {1|2|5|6|7}, it represents {CH1|CH2|MATH|REFA|REFB} respectively.

- **Return format**

  Query returns 1 or 0.

- **For example**

:CHAN1:SELect	Select CH1
:CHAN1:SELect?	Query returns 1, it represents that CH1 is selected.

TIMebase Command

This command is to change the horizontal scale (timebase) of the current channel and to generate the horizontal position (trigger offset) in memory. Changing the horizontal scale may cause the waveform extended or compressed relative to the center of the screen; changing the horizontal position may cause waveform position away from the center of screen.

:TIMebase:MODE

- **Command format**

  :TIMebase:MODE {MAIN | WINDow}

  :TIMebase:MODE?

- **Functional description**

  Set the timebase mode, MAIN or WINDow (<Zoomed> timebase).

- **Return format**

  Query returns MAIN or WINDow.

- **For example**

:TIM:MODE MAIN	Set the timebase mode to MAIN
:TIM:MODE?	Query returns MAIN

:TIMebase:OFFSet

- **Command format**

  :TIMebase:OFFSet <offset>

:TIMebase:OFFSet?

- **Functional description**

  Adjust the offset of MAIN timebase, which is the offset of waveform position relative to the center of the screen.

- **Return format**

  Query returns <offset> by scientific notation and the unit is s.

- **For example**

:TIM:OFFS 1s	Set the offset of MAIN to 1s
:TIM:OFFS?	Query returns 1.000e000

:TIMebase:WINDow:OFFSet

- **Command format**

  :TIMebase:WINDow:OFFSet <offset>

  :TIMebase:WINDow:OFFSet?

- **Functional description**

  Adjust the offset of WINDow timebase, which is the offset of waveform position relative to the center of the screen.

- **Return format**

  Query returns <offset> by scientific notation and the unit is s.

- **For example**

:TIM:WIND:OFFS 1s	Set the offset of WINDow to 1s
:TIM:WIND:OFFS?	Query returns 1.000e000

:TIMebase:SCALe

- **Command format**

  :TIMebase:SCALe {<scale> | UP | DOWN}

  :TIMebase:SCALe?

- **Functional description**

  Set the scale for MAIN timebase, which is s/div.

  <scale>: timebase value

  UP: increase one scale based on the current scale

  DOWN : decrease one scale based on the current scale

- **Return format**

  Query returns < scale> by scientific notation and the unit is s/div.

- **For example**

:TIM:SCAL 2s	Set the offset of MAIN timebase to 2 s/div
:TIM:SCAL?	Query returns 2.000e000

:TIMebase:WINDow:SCALe

- **Command format**

  :TIMebase:WINDow:SCALe < scale >

  :TIMebase:WINDow:SCALe?

- **Functional description**

  Set the scale for WINDow (<Zoomed>) timebase, which is s/div.

- **Return format**

  Query returns < scale> by scientific notation and the unit is s/div.

- **For example**

:TIM:WIND:SCAL 2s	Set the offset of WINDow timebase to 2 s/div
:TIM:WIND:SCAL?	Query returns 2.000e000

:TIMebase:HOLDoff

- **Command format**

  :TIMebase:HOLDoff <time>

  :TIMebase:HOLDoff?

- **Functional description**

  Set trigger holdoff time, the range is 100ns~10s.

- **Return format**

  Query returns trigger holdoff time by scientific notation and the unit is s.

- **For example**

:TIM:HOLD 1s	Set trigger holdoff time to 1s
:TIM:HOLD?	Query returns 1.000e000

:TIMebase:INDPendent

- **Command format**

  :TIMebase:INDPendent { {1|ON} | {0|OFF} }

  :TIMebase:INDPendent?

- **Functional description**

  Turn on/off independent timebase mode.

- **Return format**

  Query returns 1 or 0, it represents ON or OFF respectively.

- **For example**

  :TIMebase:INDPendent ON Turn on independent timebase mode

  :TIMebase:INDPendent? Query returns 1

:TIMebase:SPLit:SCReen

- **Command format**

  :TIMebase:SPLit:SCReen { {1|ON} | {0|OFF} }

  :TIMebase:SPLit:SCReen?

- **Functional description**

  Turn on/off the split screen display in independent timebase.

- **Return format**

  Query returns 1 or 0, it represents ON or OFF respectively.

- **For example**

  :TIMebase:SPLit:SCReen ON Turn on split screen display

  :TIMebase:SPLit:SCReen? Query returns 1

FUNCtion Command

This command is used to display the operation result of the waveform of CH1 and CH2.The operation is add, subtract, multiply, divide, AND, OR, NOT, XOR and FFT. Set filter to use the expression to operating.

:FUNCtion:MATH:MODE

- **Command format**

  FUNCtion:MATH:MODE {MATH|FFT|LOGic|FILTer|ADVance}

  FUNCtion:MATH:MODE?

- **Functional description**

  Select MATH mode.

- **Return format**

  Query returns {MATH|FFT|LOGic|FILTer|ADVance}.

- **For example**

  FUNC:MATH:MODE FFT Select MATH mode to FFT

  FUNC:MATH:MODE? Query returns FFT

:FUNCtion:OPERation

- **Command format**

  :FUNCtion:OPERation {ADD | SUBTract | MULTiply | DIVide | AND | OR | NOT | XOR }

  :FUNCtion:OPERation?

- **Functional description**

  Set the functional operator, which includes the basic and logical operation. That is add, subtract, multiply and divide, AND, OR, NOT, XOR

- **Return format**

  Query returns {ADD | SUBTract | MULTiply | DIVide | AND | OR | NOT | XOR }.

- **For example**

:FUNCtion:OPERation ADD	Use add operator: src1+src2
:FUNCtion:OPERation?	Query returns ADD

:FUNCtion:DIGital<n>:THReshold

- **Command format**

  :FUNCtion:DIGital<n>:THReshold <value>

  :FUNCtion:DIGital<n>:THReshold?

- **Functional description**

  Set the logic threshold for the specified channel, if it greater than the threshold, it takes 1, if it less than the threshold, it takes 0, n take value as 1, 2.

- **Return format**

  Query returns 1.000e000, and the unit is V.

- **For example**

:FUNC:DIG1:THR 1V	Set the logic threshold of CH1 to 1 V
:FUNC:DIG1:THR?	Query returns 1.000e000

:FUNCtion:SOURce<m>

- **Command format**

  :FUNCtion:SOURce<m> {CHANnel1| CHANnel2}

  :FUNCtion:SOURce<m>?

- **Functional description**

  SOURce <m> represents source 1 or source 2, <m> take value as 1, 2.

  SOURce1 is used to select the first source of operator mathematical function, and it can be a single source for Filter, FFT.

  SOURce2 is used to select the second source of operator mathematical function. Single source

of Filter and FFT are not suitable.

<value> represents CHANnel<n>, <n> take value as 1/2{CH1/ CH2}.

- **Return format**

Query returns CHANnel1, CHANnel2.

- **For example**

:FUNCtion:SOUR1 CHAN1	Set CH1 as the first source
:FUNCtion:SOUR1?	Query returns CHANnel1
:FUNCtion:SOUR2 CHAN2	Set CH2 as the second source
:FUNCtion:SOUR2?	Query returns CHANnel2
:FUNCtion:OPERation ADD	Add source 1 and source 2 channel

:FUNCtion:FFT:WINDow

- **Command format**

:FUNCtion:FFT:WINDow {RECTangular|HANNing|HAMMing|BMAN|FLATtop}

:FUNCtion:FFT:WINDow?

- **Functional description**

FFT add window to intercept signal. RECT, HANN, HAMM, BMAN, FLAT is respectively rectangular window, Hanning window, Hamming window, Blacman window and Flat top window.

- **Return format**

Query returns { RECTangular|HANNing|HAMMing|BMAN|FLATtop }.

- **For example**

:FUNCtion:SOUR1 CHAN1	Set CH1 as the source
:FUNC:FFT:WIND HAMM	Add Hamming window
:FUNC:FFT:WIND?	Query returns HAMMing

:FUNCtion:FFT:DISPlay

- **Command format**

:FUNCtion:FFT:DISPlay {FULL| SPLIt| WATerfall1| WATerfall2| INDependent}

:FUNCtion:FFT:DISPlay?

- **Functional description**

Set display mode for FFT.

- **Return format**

Query returns {FULL| SPLIt| WATerfall1| WATerfall2| INDependent }.

FULL is full display, SPLIt is split display, INDPendent represents independent display FFT

frequency spectrum, WATerfall1 is waterfall 1 curve, WATerfall2 is waterfall 2 curve.

- **For example**

:FUNCtion:FFT:DISPlay FULL Set the display mode of FFT to FULL

:FUNCtion:FFT:DISPlay? Return FULL

:FUNCtion:FFT:WATerfall:SLICe

- **Command format**

:FUNCtion:FFT:WATerfall:SLICe <value>

:FUNCtion:FFT:WATerfall:SLICe ?

- **Functional description**

Select the segment of FFT waterfall curve, the selection range is 1~200.

- **Return format**

Query returns the selected segments.

- **For example**

:FUNCtion:FFT:WATerfall:SLICe 100 Select 100th frame of FFT waterfall curve

:FUNCtion:FFT:WATerfall:SLICe ? Query returns 100

:FUNCtion:FFT:POINts

- **Command format**

:FUNCtion:FFT:POINts {8K| 16K| 32K| 64K| 128K | 256K | 512K | 1M}

:FUNCtion:FFT:POINts?

- **Functional description**

Set the point for FFT.

- **Return format**

Query returns {8K| 16K| 32K| 64K| 128K | 256K | 512K | 1M}.

- **For example**

:FUNCtion:FFT:POINts 8K Set FFT point to 8K

:FUNCtion:FFT:POINts? Return 8K

:FUNCtion:FFT:VTYPe

- **Command format**

:FUNCtion:FFT:VTYPe {VRMS|DBRMS}

:FUNCtion:FFT:VTYPe?

- **Functional description**

Select the unit of FFT on vertical direction to dBRMS or VRMS.

dBRMS represents power RMS, VRMS represents current RMS.

- **Return format**

  Query returns {VRMS|DBRMS}。

- **For example**

  :FUNCtion:SOUR1 CHAN1 Set CH1 as the source

  :FUNC:FFT:VTYP VRMS Set the unit of FFT on vertical direction to VRMS

  :FUNC:FFT:VTYP? Query returns VRMS

:FUNCtion:FFT:FREQuency

- **Command format**

  :FUNCtion:FFT:FREQuency?

- **Functional description**

  Acquire the center frequency of frequency spectrum waveform after FFT.

- **Return format**

  Query returns the center frequency of frequency spectrum waveform, the unit is Hz.

- **For example**

  :FUNCtion:FFT:FREQuency? Query returns 1.000e003

:FUNCtion:FFT:FREQuency:STARt

- **Command format**

  :FUNCtion:FFT:FREQuency:STARt <freq>

  :FUNCtion:FFT:FREQuency:STARt?

- **Functional description**

  Set the start frequency of FFT.

- **Return format**

  Query returns 1.000e003, the unit is Hz.

- **For example**

  :FUNCtion:FFT:FREQuency:STARt 1KHz Set the start frequency to 1 KHz

  :FUNC:FFT:FREQ:STARt? Query returns 1.000e003

:FUNCtion:FFT:FREQuency:END

- **Command format**

  :FUNCtion:FFT:FREQuency:END <freq>

  :FUNCtion:FFT:FREQuency:END?

- **Functional description**

Set the stop frequency of FFT.

- **Return format**

Query returns 1.000e003, the unit is Hz.

- **For example**

:FUNCtion:FFT:FREQuency:END 1KHz	Set the stop frequency to 1 KHz
:FUNC:FFT:FREQ:END?	Query returns 1.000e003

:FUNCtion:FFT:FREQuency:CENTer

- **Command format**

:FUNCtion:FFT:FREQuency:CENTer <freq>

:FUNCtion:FFT:FREQuency:CENTer?

- **Functional description**

Set the center frequency of FFT.

- **Return format**

Query returns 1.000e003, the unit is Hz.

- **For example**

:FUNCtion:FFT:FREQuency:CENTer 1KHz	Set the center frequency to 1 KHz
:FUNC:FFT:FREQ:CENTer?	Query returns 1.000e003

:FUNCtion:FFT:FREQuency:BW

- **Command format**

:FUNCtion:FFT:FREQuency:BW <freq>

:FUNCtion:FFT:FREQuency:BW?

- **Functional description**

Set the frequency bandwidth of FFT.

- **Return format**

Query returns 1.000e003, the unit is Hz.

- **For example**

:FUNCtion:FFT:FREQuency:BW 1KHz	Set the frequency bandwidth to 1 KHz
:FUNC:FFT:FREQ:BW?	Query returns 1.000e003

:FUNCtion:FFT:FREQuency:TRACk

- **Command format**

:FUNCtion:FFT:FREQuency:TRACk {{1 | ON} | {0 | OFF}}

:FUNCtion:FFT:FREQuency:TRACk?

- **Functional description**

  Set the frequency trace switch of FFT.

- **Return format**

  Query returns the status of frequency trace, 0 represents not trace, 1 represents trace.

- **For example**

  :FUNCtion:FFT:FREQuency:TRACk ON Turn on frequency trace

  :FUNCtion:FFT:FREQuency:TRACk? Query returns 1

:FUNCtion:FFT:DETEction:REALTime

- **Command format**

  :FUNCtion:FFT:DETEction:REALTime {PPEAK|NPEAK| AVERage| SAMPle}

  :FUNCtion:FFT:DETEction:REALTime?

- **Functional description**

  Set the detection mode of real-time frequency spectrum.

  PPEAK: take the maximum value within the range of each sampling point

  NPEAK: take the minimum value within the range of each sampling point

  AVERage: take the average value within the range of each sampling point

  SAMPle: take the first point within the range of each sampling point

- **Return format**

  Query returns the detection mode of real-time frequency spectrum.

- **For example**

  :FUNCtion:FFT:DETEction:REALTime PPEAK

  Set the detection mode of real-time frequency spectrum to +peak detection

  :FUNCtion:FFT:DETEction:REALTime? Query returns PPEAK

:FUNCtion:FFT:DETEction:AVERage

- **Command format**

  :FUNCtion:FFT:DETEction:AVERage {OFF|PPEAK|NPEAK| AVERage|SAMPle}

  :FUNCtion:FFT:DETEction:AVERage?

- **Functional description**

  Set the detection mode of average frequency spectrum.

  OFF: turn off the average frequency spectrum.

  PPEAK: take the maximum value within the range of each sampling point

  NPEAK: take the minimum value within the range of each sampling point

  AVERage: take the average value within the range of each sampling point

SAMPle: take the first point within the range of each sampling point

- **Return format**

Query returns the detection mode of average frequency spectrum.

- **For example**

:FUNCtion:FFT:DETEction:AVERage PPEAK

Set the detection mode of average frequency spectrum to +peak detection

:FUNCtion:FFT:DETEction:AVERage? Query returns PPEAK

:FUNCtion:FFT:DETEction:AVERage:COUNt

- **Command format**

:FUNCtion:FFT:DETEction:AVERage:COUNt <value>

:FUNCtion:FFT:DETEction:AVERage:COUNt?

- **Functional description**

Set the average count of average frequency spectrum, the range is 2~512.

- **Return format**

Query the average count of average frequency spectrum.

- **For example**

:FUNCtion:FFT:DETEction:AVERage:COUNt 56

Set the average count of average frequency spectrum to 56

:FUNCtion:FFT:DETEction:AVERage:COUNt? Query returns 56

:FUNCtion:FFT:DETEction:MAXHold

- **Command format**

:FUNCtion:FFT:DETEction:MAXHold {OFF|PPEAK|NPEAK|AVERage|SAMPle}

:FUNCtion:FFT:DETEction:MAXHold?

- **Functional description**

Set the detection mode of maximum hold frequency spectrum.

OFF: turn off the average frequency spectrum.

PPEAK: take the maximum value within the range of each sampling point

NPEAK: take the minimum value within the range of each sampling point

AVERage: take the average value within the range of each sampling point

SAMPle: take the first point within the range of each sampling point

- **Return format**

Query returns the detection mode of maximum hold frequency spectrum.

- **For example**

:FUNCtion:FFT:DETEction:MAXHold PPEAK

Set the detection mode of maximum hold frequency spectrum to +peak detection

:FUNCtion:FFT:DETEction:MAXHold?

Query returns PPEAK

:FUNCtion:FFT:DETEction:MINHold

- **Command format**

:FUNCtion:FFT:DETEction:MINHold {OFF|PPEAK|NPEAK| AVERage|SAMPle}

:FUNCtion:FFT:DETEction:MINHold?

- **Functional description**

Set the detection mode of minimum hold frequency spectrum.

OFF: turn off the average frequency spectrum.

PPEAK: take the maximum value within the range of each sampling point

NPEAK: take the minimum value within the range of each sampling point

AVERage: take the average value within the range of each sampling point

SAMPle: take the first point within the range of each sampling point

- **Return format**

Query returns the detection mode of minimum hold frequency spectrum.

- **For example**

:FUNCtion:FFT:DETEction:MINHold PPEAK

Set the detection mode of frequency spectrum to +peak detection

:FUNCtion:FFT:DETEction:MINHold? Query returns PPEAK

:FUNCtion:FFT:DETEction:RESet

- **Command format**

:FUNCtion:FFT:DETEction:RESet

- **Functional description**

Reset the average, maximum hold and minimum hold frequency spectrum.

- **For example**

:FUNCtion:FFT:DETEction:RESet Reset each frequency spectrum

:FUNCtion:FFT:MARK:TYPE

- **Command format**

:FUNCtion:FFT:MARK:TYPE {OFF|AUTO|THReshold| MANUal}

:FUNCtion:FFT:MARK:TYPE?

- **Functional description**

  Set mark type of frequency spectrum.

  OFF: turn off the marker of frequency spectrum

  AUTO: automatically mark

  THReshold: threshold mark

  MANUal: mark by manual

- **Return format**

  Query returns the selected mark type.

- **For example**

  :FUNCtion:FFT:MARK:TYPE AUTO Set the mark type to AUTO

  :FUNCtion:FFT:MARK:TYPE? Query returns AUTO

:FUNCtion:FFT:MARK:SOURce

- **Command format**

  :FUNCtion:FFT:MARK:SOURce {REALtime|AVERage|MAXHold|MINHold}

  :FUNCtion:FFT:MARK:SOURce?

- **Functional description**

  Set the mark source of frequency spectrum's marker, this command is the common command for each frequency spectrum's marker.

  REALtime: mark the frequency spectrum in real time

  AVERage: mark the average frequency spectrum

  MAXHold: mark the maximum hold frequency spectrum

  MINHold: mark the minimum hold frequency spectrum

- **Return format**

  Query returns the selected mark source.

- **For example**

  :FUNCtion:FFT:MARK:SOURce AVERage

  Set the frequency spectrum's marker to average frequency spectrum

  :FUNCtion:FFT:MARK:SOURce? Query returns AVERage

:FUNCtion:FFT:MARK:POINts

- **Command format**

  :FUNCtion:FFT:MARK:POINts <value>

  :FUNCtion:FFT:MARK:POINts ?

- **Functional description**

Set the mark point of frequency spectrum.

<value>: the value of mark point, the range is 1~50.

- **Return format**

Query returns the mark point of frequency spectrum.

- **For example**

:FUNCtion:FFT:MARK:POINts 20

Set the mark point of frequency spectrum to 20

:FUNCtion:FFT:MARK:POINts ? Query returns 20

:FUNCtion:FFT:MARK:EVENt

- **Command format**

:FUNCtion:FFT:MARK:EVENt {1 | ON} | {0 | OFF}

:FUNCtion:FFT:MARK:EVENt?

- **Functional description**

Turn on/off the mark list of frequency spectrum's marker.

- **Return format**

Query the status of the mark list, 1 represents ON, 0 represents OFF.

- **For example**

:FUNCtion:FFT:MARK:EVENt ON Turn on the mark list

:FUNCtion:FFT:MARK:EVENt? Query returns 1

:FUNCtion:FFT:MARK:DATA?

- **Command format**

:FUNCtion:FFT:MARK:DATA?

- **Functional description**

Read the mark event data under FFT.

- **Return format**

Query returns the mark event data under FFT, returned data is conform with Appendix 2:IEEE 488.2 Binary System Data Format.

- **For example**

:FUNCtion:FFT:MARK:DATA?Query returns :

#9000000089FFT,

ID,Freq,Amp,

1,1.000E+003,7.800E-001,

2,2.000E+003,7.900E-001,

3,3.000E+003,7.700E-001,

4,4.000E+003,7.300E-001,

5,5.000E+003,7.400E-001,

FFT represents FFT mode, the event table data in CSV format followed behind. The specified format of the event table data is automatically adapted by different devices. The data are separated by commas and will automatically line wrap according to the decoding list.

:FUNCtion:FFT:MARK:THReshold:LEVel

- **Command format**

  :FUNCtion:FFT:MARK:THReshold:LEVel <value>

  :FUNCtion:FFT:MARK:THReshold:LEVel?

- **Functional description**

  Set the threshold voltage for threshold spectrum's marker

  <value>: threshold voltage, when the unit of vertical is Vrms, the unit is V; when the unit of vertical is dBVrms, the unit is dB; if the unit is not match with the unit of vertical, then this setting is invalid.

- **Return format**

  Query returns the threshold of spectrum's marker by scientific notation, the unit is related with the command :FUNCtion:FFT:VTYPe.

- **For example**

  :FUNCtion:FFT:MARK:THReshold:LEVel -12.5dB

  Set the threshold of spectrum's marker to -12.5dB

  :FUNCtion:FFT:MARK:THReshold:LEVel? Query returns -1.250e-001

  :FUNCtion:FFT:MARK:THReshold:LEVel 0.15V

  Set the threshold of spectrum's marker to 0.15V

  :FUNCtion:FFT:MARK:THReshold:LEVel? Query returns 1.500e-001

:FUNCtion:FFT:MARK:MANUal:PEAK

- **Command format**

  :FUNCtion:FFT:MARK:MANUal:PEAK

- **Functional description**

  Move the marker to the maximum peak.

- **For example**

  :FUNCtion:FFT:MARK:MANUal:PEAK

  Move the marker to the maximum peak

:FUNCtion:FILTer:TYPE

- **Command format**

  :FUNCtion:FILTer:TYPE {LP|HP|BP|BS}

  :FUNCtion:FILTer:TYPE?

- **Functional description**

  Set the filter type, LP, HP, BP, BS represents low-pass filter, high-pass filter, band-pass filter and band-limit filter respectively.

- **Return format**

  Query returns LP, HP, BP, BS.

- **For example**

  :FUNCtion:SOUR1 CHAN1 Set CH1 as the source

  :FUNC:FILT:TYPE BP Set the filter type to BP

  :FUNC:FILT:TYPE? Query returns BP

:FUNCtion:FILTer:FREQuency:HIGH

- **Command format**

  :FUNCtion:FILTer:FREQuency:HIGH < freq>

  :FUNCtion:FILTer:FREQuency:HIGH?

- **Functional description**

  Set the cut-off frequency for the upper limit of filter. It is suitable for high-pass filter, band-pass filter and band-limit filter.

- **Return format**

  Query returns 1.000e003, the unit is Hz.

- **For example**

  :FUNCtion:SOUR1 CHAN1 Set CH1 as the source

  :FUNC:FILT:FREQ:HIGH 1KHz

  Set the upper limit of filter to the cut-off frequency of 1 KHz

  :FUNC:FILT:FREQ:HIGH? Query returns 1.000e003

:FUNCtion:FILTer:FREQuency:LOW

- **Command format**

  :FUNCtion:FILTer:FREQuency:LOW <freq>

  :FUNCtion:FILTer:FREQuency:LOW?

- **Functional description**

  Set the cut-off frequency for the lower limit of filter. It is suitable for low-pass filter, band-pass

filter and band-limit filter.

- **Return format**

  Query returns 6.000e001, the unit is Hz.

- **For example**

  :FUNC:SOUR1 CHAN1 Set CH1 as the source

  :FUNC:FILT:FREQ:LOW 60Hz

  Set the lower limit of filter to the cut-off frequency of 60 Hz

  :FUNC:FILT:FREQ:LOW? Query returns 6.000e001

:FUNCtion:LOGic:INVert

- **Command format**

  :FUNCtion:LOGic:INVert {{1|ON}|{0|OFF}}

  :FUNCtion:LOGic:INVert?

- **Functional description**

  Turn on/off logical reversed phase.

- **Return format**

  Query returns 1 or 0, it represents ON or OFF respectively.

- **For example**

  :FUNCtion:LOGic:INVert OFF Turn off reversed phase

  :FUNCtion:LOGic:INVert?

  Query returns 0, it represents that logical reversed phase is closed.

:FUNCtion:EXPRession

- **Command format**

  :FUNCtion:EXPRession <expression>

- **Functional description**

  Use free combination expression to perform mathematical calculation.

  Expression format refer to Advance option in MATH menu of the oscilloscope, <expression> is belong to ASCII character string.

- **For example**

  :FUNCtion:EXPRession "CH1*CH2" Multiply CH1 with CH2

MEASure Command

The command is for the basic measurement operation, all parameter measurement can get the test value without open measurement function; by default, and it will turn on measurement and acquire the test value automatically. In general, test result is returned in scientific notation.

:MEASure:ALL

- **Command format**

  :MEASure:ALL {{1 | ON} | {0 | OFF}}

  :MEASure:ALL?

- **Functional description**

  Turn on/off all measurement function.

- **Return format**

  Query returns whether all measurement function is turned on.

- **For example**

:MEASure:ALL ON	Turn on all measurement function
:MEASure:ALL?	Query returns 1

:MEASure:CLEar

- **Command format**

  :MEASure:CLEar

- **Functional description**

  Clear the currently measured parameter and refresh the measured data.

- **For example**

:MEAS:CLE	Clear the currently measured parameter

:MEASure:SOURce

- **Command format**

  :MEASure:SOURce <source>

  :MEASure:SOURce?

- **Functional description**

  Select the measuring source, <source> is { CHANnel<n> | MATH }, n take value as 1, 2.

- **Return format**

  Query returns {CHANnel1 | CHANnel2 | MATH }.

- **For example**

:MEAS:SOUR CHAN1	Select CH1 as the measuring source

:MEAS:SOUR? Return CHANnel1

:MEASure:SLAVe:SOURce

- **Command format**

  :MEASure:SLAVe:SOURce <source>

  :MEASure:SLAVe:SOURce?

- **Functional description**

  Select the measuring slave source, <source> is { CHANnel<n> | MATH }, n take value as1, 2.

- **Return format**

  Query returns {CHANnel1 | CHANnel2 | MATH }.

- **For example**

  :MEAS:SLAV:SOUR CHAN1 Select CH1 as the measuring slave source

  :MEAS:SLAV:SOUR? Return CHANnel1

:MEASure:STATistic:DISPlay

- **Command format**

  :MEASure:STATistic:DISPlay { {1|ON} | {0|OFF} }

  :MEASure:STATistic:DISPlay?

- **Functional description**

  Turn on/off statistical function.

- **Return format**

  Query returns 1 or 0, it represents ON or OFF respectively.

- **For example**

  :MEASure:STATistic:DISPlay ON Turn on statistical function

  :MEASure:STATistic:DISPlay? Query returns 1

:MEASure:STATistic:RESet

- **Command format**

  :MEASure:STATistic:RESet

- **Functional description**

  Clear historical statistics data and restart to statistic.

- **For example**

  :MEASure:STATistic:RESet

  Clear historical statistics data and restart to statistic

:MEASure:STATistic:ITEM

- **Command format**

  :MEASure:STATistic:ITEM <item>

  :MEASure:STATistic:ITEM? <type>,<item>

- **Functional description**

  Turn on the statistical function of specified arbitrary waveform and query the statistical result of arbitrary waveform.

  <item> represents waveform parameter:

  {VMAX|VMIN|VPP|VTOP|VBASe|VAMPlitude|VAVerage|CYCVAVerage|VMIDdle|VRMS|ACRMS|CYCRMS|AREa|CYCAREa|FREQuency|PHASe|PERiod|OVERshoot|PREShoot|RISetime|FALLtime|PWIDth|NWIDth|PDUTy|NDUTy|PDELay|NDELay|PULSes|FRR|FRF|FFR|FFF|LRR|LRF|LFR|LFF}.

  <type> represents statistical type:

  {MAXimum|MINimum|CURRent|AVERages|DEViation}, it represents maximum, minimum, the current value, average value and deviation respectively.

- **Return format**

  Query returns the statistic result by scientific notation.

- **For example**

  :MEASure:STATistic:ITEM CYCVAV

  Turn on the statistical function of periodical average

  :MEASure:STATistic:ITEM? MAX,CYCVAV

  Return the maximum 1.120e000 of periodical average

:MEASure:WINDow

- **Command format**

  :MEASure:WINDow {SCReen | CURSor}

  :MEASure:WINDow?

- **Functional description**

  Set the window area to be measured. SCReen is full screen, CURSor is cursor screen.

  When the measuring area is cursor area, use the command :CURSor:CURAX

  and :CURSor:CURBX to adjust the vertical range of the cursor line.

- **Return format**

  Query returns {SCReen | CURSor}

- **For example**

  :MEASure:WINDow CURSor Set the measuring area to cursor area

:MEASure:WINDow? Query returns CURSor

:MEASure:THReshold:DEFault

- **Command format**

  :MEASure:THReshold:DEFault <source>

- **Functional description**

  Set the default value of threshold, the lower threshold is 10%, the middle threshold is 50%, and the upper threshold is 90%. <source> represents {CHANnel1| CHANnel2| MATH}.

- **For example**

  :MEASure:THReshold:DEFault CHANnel1

  Set CH1's threshold to default value

:MEASure:THReshold

- **Command format**

  :MEASure:THReshold <source> [,<lower>][,< middle>][,<upper>]

  :MEASure:THReshold? <source>

- **Functional description**

  Set to measure the upper, middle, lower limit of <source> under the user-defined mode, it can be set separately. The threshold will affect the measurement of time, delay, phase and duty cycle.

  <source> represents {CHANnel1| CHANnel2| MATH}.

  <lower> take value as 5% - 93%; <middle> take value as 6% - 94%;<upper> take value as 7% - 95%.

- **Return format**

  Query returns the measuring threshold of the upper, middle and lower limit by scientific notation and the unit is %.

- **For example**

  :MEASure:THReshold CHANnel1,20,40,80

  Set CH1 to measure the upper, middle, lower limit of threshold

  :MEASure:THReshold? CHANnel1

  Query returns 2.000e001, 4.000e001, 8.000e001

  :MEASure:THReshold CHANnel1,,40

  Set CH1 to measure the middle limit of threshold 40%

:MEASure:PDUTy?

- **Command format**

  :MEASure:PDUTy? [<source>]

- **Functional description**

  Set to measure the positive duty cycle of specified channel's waveform. <source> take value as CHANnel1, CHANnel2 or MATH. If it is omitted, it represents the current channel.

- **Return format**

  Query returns 5.000e001, the unit is %.

:MEASure:NDUTy?

- **Command format**

  :MEASure:NDUTy? [<source>]

- **Functional description**

  Set to measure the negative duty cycle of specified channel's waveform. <source> take value as CHANnel1, CHANnel2 or MATH. If it is omitted, it represents the current channel.

- **Return format**

  Query returns 5.000e001, the unit is %.

:MEASure:PDELay?

- **Command format**

  :MEASure:PDELay? [<source1>,<source2>]

- **Functional description**

  Set to measure time delay 用于测量<source1>, <source2>相对于上升沿的时间延迟。<source> take value as CHANnel1, CHANnel2 or MATH.

- **Return format**

  Query returns -1.000e-004, the unit is s.

- **For example**

  Measuring the time delay with respect to the rising edge

  :MEASure:PDEL? CHAN1,CHAN2

:MEASure:NDELay?

- **Command format**

  :MEASure:NDELay? [<source1>,<source2>]

- **Functional description**

  Measuring the time delay of <source1>, <source2> with respect to the falling edge. <source>

take value as CHANnel1, CHANnel2 or MATH.

- **Return format**

  Query returns -1.000e-004, the unit is s.

- **For example**

  Measuring the time delay with respect to the falling edge

  :MEASure:NDEL? CHAN1,CHAN2

:MEASure:PHASe?

- **Command format**

  :MEASure:PHASe? [<source1>,<source2>]

- **Functional description**

  Timed measurement of the amount of time <source1> is ahead or behind with respect to <source2>, and expressed in degree. 360° is one period. <source> take value as CHANnel1, CHANnel2 or MATH.

- **Return format**

  Query returns 1.000e001, the unit is ° (degree).

- **For example**

  Measuring the amount of time <source1> is ahead or behind with respect to <source2>

  :MEASure:PHAS? CHAN1,CHAN2

:MEASure:VPP?

- **Command format**

  :MEASure:VPP? [<source>]

- **Functional description**

  Set to measure the peak-to-peak of the specified channel's waveform. <source> take value as CHANnel1, CHANnel2 or MATH. If it is omitted, it represents the current channel.

- **Return format**

  Query returns 3.120e000, the unit is V.

:MEASure:VMAX?

- **Command format**

  :MEASure:VMAX? [<source>]

- **Functional description**

  Set to measure the maximum of the specified channel's waveform. <source> take value as CHANnel1, CHANnel2 or MATH. If it is omitted, it represents the current channel.

- **Return format**

  Query returns 2.120e000, the unit is V.

:MEASure:VMIN?

- **Command format**

  :MEASure:VMIN? [<source>]

- **Functional description**

  Set to measure the minimum of the specified channel's waveform. <source> take value as CHANnel1, CHANnel2 or MATH. If it is omitted, it represents the current channel.

- **Return format**

  Query returns -2.120e000, the unit is V.

:MEASure:VAMPlitude?

- **Command format**

  :MEASure:VAMPlitude? [<source>]

- **Functional description**

  Set to measure the amplitude of the specified channel's waveform. <source> take value as CHANnel1, CHANnel2 or MATH. If it is omitted, it represents the current channel.

- **Return format**

  Query returns 3.120e000, the unit is V.

:MEASure:VTOP?

- **Command format**

  :MEASure:VTOP? [<source>]

- **Functional description**

  Set to measure the top value of the specified channel's waveform. <source> take value as CHANnel1, CHANnel2 or MATH. If it is omitted, it represents the current channel.

- **Return format**

  Query returns 3.120e000, the unit is V.

:MEASure:VBASe?

- **Command format**

  :MEASure:VBASe? [<source>]

- **Functional description**

  Set to measure the bottom value of the specified channel's waveform. <source> take value as CHANnel1, CHANnel2 or MATH. If it is omitted, it represents the current channel.

- **Return format**

  Query returns -3.120e000, the unit is V.

:MEASure:VMIDdle?

- **Command format**

  :MEASure:VMIDdle? [<source>]

- **Functional description**

  Set to measure the middle value of the specified channel's waveform. <source> take value as CHANnel1, CHANnel2 or MATH. If it is omitted, it represents the current channel.

- **Return format**

  Query returns 0.120e000, the unit is V.

:MEASure:VAVerage?

- **Command format**

  :MEASure:VAVerage? [<interval>][,<source>]

- **Functional description**

  Set to measure the average value of the specified channel's waveform. <source> is the specified channel, it can take value as CHANnel1, CHANnel2 or MATH. If <source> is not assigned, then the current channel will assign to be the specified channel by default; <interval> is measuring interval, it can take value as CYCLe, DISPlay. CYCLe represents integer cycle period, DISPlay represents full screen, if <interval> is not assigned, select DISPlay by default.

- **Return format**

  Query returns 1.120e000, the unit is V.

:MEASure:VRMS?

- **Command format**

  :MEASure:VRMS? [<interval>][,<source>]

- **Functional description**

  Measure RMS (root mean square) of the specified channel's waveform. <source> is the specified channel, it can take value as CHANnel1, CHANnel2 or MATH. If <source> is not assigned, then the current channel will assign to be the specified channel by default; <interval> is measuring interval, it can take value as CYCLe, DISPlay. CYCLe represents integer cycle period, DISPlay represents full screen, if <interval> is not assigned, select DISPlay by default.

- **Return format**

  Query returns 1.230e000, the unit is V.

:MEASure:ACRMs?

- **Command format**

  :MEASure:ACRMs? [<source>]

- **Functional description**

  Measure AC RMS (root mean square) of the specified channel's waveform. <source> is the specified channel, it can take value as CHANnel1, CHANnel2 or MATH. If <source> is not assigned, then the current channel will assign to be the specified channel by default.

- **Return format**

  Query returns 1.230e000, the unit is V.

:MEASure:AREa?

- **Command format**

  :MEASure:AREa? [<interval>][,<source>]

- **Functional description**

  Measure the area of the specified channel's waveform. <source> is the specified channel, it can take value as CHANnel1, CHANnel2 or MATH. If <source> is not assigned, then the current channel will assign to be the specified channel by default; <interval> is measuring interval, it can take value as CYCLe, DISPlay. CYCLe represents integer cycle period, DISPlay represents full screen, if <interval> is not assigned, select DISPlay by default.

- **Return format**

  Query returns 3.456e002, the unit is Vs.

:MEASure:OVERshoot?

- **Command format**

  :MEASure:OVERshoot? [<source>]

- **Functional description**

  Measure the overshoot of the specified channel's waveform. <source> take value as CHANnel1, CHANnel2 or MATH. If it is omitted, it represents the current channel.

- **Return format**

  Query returns 1.230e-002, the unit is %.

:MEASure:PREShoot?

- **Command format**

  :MEASure:PREShoot? [<source>]

- **Functional description**

Measure the preshoot of the specified channel's waveform. <source> take value as CHANnel1, CHANnel2 or MATH. If it is omitted, it represents the current channel.

- **Return format**

Query returns 1.230e-002, the unit is %.

:MEASure:FREQuency?

- **Command format**

:MEASure:FREQuency? [<source>]

- **Functional description**

Measure the frequency of the specified channel's waveform. <source> take value as CHANnel1, CHANnel2 or MATH. If it is omitted, it represents the current channel.

- **Return format**

Query returns 2.000e003, and the unit is Hz.

:MEASure:RISetime?

- **Command format**

:MEASure:RISetime? [<source>]

- **Functional description**

Measure the rising time of the specified channel's waveform. <source> take value as CHANnel1, CHANnel2 or MATH. If it is omitted, it represents the current channel.

- **Return format**

Query returns 5.000e-005, and the unit is s.

:MEASure:FALLtime?

- **Command format**

:MEASure:FALLtime? [<source>]

- **Functional description**

Measure the falling time of the specified channel's waveform. <source> take value as CHANnel1, CHANnel2 or MATH. If it is omitted, it represents the current channel.

- **Return format**

Query returns 5.000e-005, and the unit is s.

:MEASure:PERiod?

- **Command format**

:MEASure:PERiod? [<source>]

- **Functional description**

Measure the period of the specified channel's waveform. <source> take value as CHANnel1, CHANnel2 or MATH. If it is omitted, it represents the current channel.

- **Return format**

Query returns 5.000e-003, and the unit is s.

:MEASure:PWIDth?

- **Command format**

:MEASure:PWIDth? [<source>]

- **Functional description**

Measure the positive pulse width of the specified channel's waveform. <source> take value as CHANnel1, CHANnel2 or MATH. If it is omitted, it represents the current channel.

- **Return format**

Query returns 5.000e-003, and the unit is s.

:MEASure:NWIDth?

- **Command format**

:MEASure:NWIDth? [<source>]

- **Functional description**

Measure the negative pulse width of the specified channel's waveform. <source> take value as CHANnel1, CHANnel2 or MATH. If it is omitted, it represents the current channel.

- **Return format**

Query returns 5.000e-003, and the unit is s.

:MEASure:PULSes?

- **Command format**

:MEASure:PULSes? [<source>]

- **Functional description**

Measure the positive pulse count of the specified channel's waveform. <source> take value as CHANnel1, CHANnel2 or MATH. If it is omitted, it represents the current channel.

- **Return format**

Query returns 2.000e+000, it represents 2 pulse strings

:MEASure:FRR?

- **Command format**

:MEASure:FRR? <source1>,<source2>

- **Functional description**

Measure the time between <source1> and the first rising edge of <source2>. <source> take value as CHANnel1, CHANnel2 or MATH.

- **Return format**

Query returns 5.000e-003，and the unit is s.

:MEASure:FRF?

- **Command format**

:MEASure:FRF? <source1>,<source2>

- **Functional description**

Measure the time between the first rising edge of <source1> and the first falling edge of <source2>. <source> take value as CHANnel1, CHANnel2 or MATH. If it is omitted, it represents the current channel.

- **Return format**

Query returns 5.000e-003，and the unit is s.

:MEASure:FFR?

- **Command format**

:MEASure:FFR? <source1>,<source2>

- **Functional description**

Measure the time between the first falling edge of <source1> and the first rising edge of <source2>. <source> take value as CHANnel1, CHANnel2 or MATH.

- **Return format**

Query returns 5.000e-003，and the unit is s.

:MEASure:FFF?

- **Command format**

:MEASure:FFF? <source1>,<source2>

- **Functional description**

Measure the time between <source1> and the first falling edge of <source2>. <source> take value as CHANnel1, CHANnel2 or MATH.

- **Return format**

Query returns 5.000e-003，and the unit is s.

:MEASure:LRR?

- **Command format**

:MEASure:LRR? <source1>,<source2>

- **Functional description**

  Measure the time between the first rising edge of <source1> and the last rising edge of <source2>. <source> take value as CHANnel1, CHANnel2 or MATH.

- **Return format**

  Query returns 5.000e-003，and the unit is s.

:MEASure:LRF?

- **Command format**

  :MEASure:LRF? <source1>,<source2>

- **Functional description**

  Measure the time between the first rising edge of <source1> and the last falling edge of <source2>. <source> take value as CHANnel1, CHANnel2 or MATH.

- **Return format**

  Query returns 5.000e-003，and the unit is s.

:MEASure:LFR?

- **Command format**

  :MEASure:LFR? <source1>,<source2>

- **Functional description**

  Measure the time between the first falling edge of <source1> and the last rising edge of <source2>. <source> take value as CHANnel1, CHANnel2 or MATH.

- **Return format**

  Query returns 5.000e-003，and the unit is s.

:MEASure:LFF?

- **Command format**

  :MEASure:LFF? <source1>,<source2>

- **Functional description**

  Measure the time between the first falling edge of <source1> and the last falling edge of <source2>. <source> take value as CHANnel1, CHANnel2 or MATH.

- **Return format**

  Query returns 5.000e-003，and the unit is s.

TRIGger Command

This command is used to control the sweep mode and specification of trigger. Trigger determines when the oscilloscope to start sampling data and display waveform.

Trigger Control

:TRIGger:MODE

- **Command format**

  :TRIGger:MODE <mode>

  :TRIGger:MODE?

- **Functional description**

  Set the trigger mode, it will automatically adjusting for different model.

  <mode> contains EDGE (edge trigger), PULSe (pulse width trigger), VIDeo (video trigger), SLOPe (slope trigger), RUNT (runt trigger), WINDow (over-amplitude trigger), DELay (delay trigger), TIMeout (timeout trigger), DURation (duration trigger), SHOLd (setup & hold trigger), NE (Nth edge trigger) , PATTern (code pattern trigger).

- **Return format**

  Query returns the trigger mode.

- **For example**

:TRIGger:MODE NE	Set the trigger mode to NE
:TRIGger:MODE?	Query returns NE

:TRIGger:FORCe

- **Command format**

  :TRIGger:FORCe

- **Functional description**

  Force the oscilloscope to generate a trigger signal so that the input waveform can be triggered and displayed when the oscilloscope does not find a suitable trigger condition.

- **For example**

:TRIG:FORC	Force trigger

:TRIGger:SWEep

- **Command format**

  :TRIGger:SWEep {AUTO|NORMal|SINGle}

  :TRIGger:SWEep?

- **Functional description**

  Select the sweep mode for trigger.

  AUTO: in no trigger condition, a trigger signal will be generated internally to force a trigger

  NORMal: it can only be generated when the trigger condition is met.

  SINGle: generating one trigger and stop when the trigger condition is met.

- **Return format**

  Query returns the sweep mode of trigger {AUTO|NORMal|SINGle}.

- **For example**

  :TRIGger:SWEep AUTO Set the trigger mode of CH1 to AUTO

  :TRIGger:SWEep? Query returns AUTO

:TRIGger:LEVel:ASETup

- **Command format**

  :TRIGger:LEVel:ASETup

- **Functional description**

  Set the trigger level at the vertical midpoint of signal amplitude.

- **For example**

  :TRIG:LEVel:ASETup Set the trigger level at the center

:TRIGger:STATus?

- **Command format**

  :TRIGger:STATus?

- **Functional description**

  Query the current status of oscilloscope's trigger.

- **Return format**

  Query returns STOP/ARMED/READY/TRIGED/AUTO/SCAN/RESET/REPLAY/WAIT.

- **For example**

  :TRIGger:STATus? Query returns AUTO

:TRIGger:LEVel

- **Command format**

  :TRIGger:LEVel <level>

  :TRIGger:LEVel?

- **Functional description**

  Set the trigger level for the normal trigger mode. The number of <level> should be set based on

the conversion of volts/div and screen information.

- **Return format**

  Query returns the set value of <level>, and the unit is V.

- **For example**

:TRIG:LEV 2	Set the trigger level of trigger to 2 V
:TRIG:LEV?	Query returns 2.000e000

:TRIGger:LEVel:LOW

- **Command format**

  :TRIGger:LEVel:LOW <level>

  :TRIGger:LEVel:LOW?

- **Functional description**

  Set the low level for the slope trigger mode. The number of <level> should be set based on the conversion of volts/div and screen information.

- **Return format**

  Query returns the set value of <level>, and the unit is V.

- **For example**

:TRIG:LEV:LOW 2	Set the trigger level of trigger to 2 V
:TRIG:LEV:LOW?	Query returns 2.000e000

:TRIGger:LEVel:HIGH

- **Command format**

  :TRIGger:LEVel:HIGH <level>

  :TRIGger:LEVel:HIGH?

- **Functional description**

  Set the high level for the slope trigger mode. The number of <level> should be set based on the conversion of volts/div and screen information.

- **Return format**

  Query returns the set value of <level>, and the unit is V.

- **For example**

:TRIG:LEV:HIGH 2	Set the trigger level of trigger to 2 V
:TRIG:LEV:HIGH?	Query returns 2.000e000

:TRIGger:SOURce

- **Command format**

:TRIGger:SOURce <source>

:TRIGger:SOURce?

- **Functional description**

Set single trigger source to input channel (CHANnel1, CHANnel2), external trigger (EXT) or AC Line. Only EDGE/ PULSe supports AC Line, EXT.

<source> represents the trigger source, it can take value as CHANnel<n>|EXT |ACLINE, <n> take value as 1, 2.

- **Return format**

Query returns the trigger source {CHANnel1|CHANnel2|EXT|ACLINE}.

- **For example**

:TRIGger:SOUR CHAN1	Set CH1 to be the trigger source
:TRIGger:SOUR?	Query returns CHANnel1

:TRIGger:COUPling

- **Command format**

:TRIGger:COUPling {DC|AC|LF|HF|NOISE}

:TRIGger:COUPling?

- **Functional description**

Set the coupling mode to DC, AC, LF(low frequency reject), HF (high frequency reject) or NOISE (noise reject).

- **Return format**

Query returns the coupling mode {DC|AC|LF|HF|NOISE}.

- **For example**

:TRIGger:COUPling AC	Set the edge trigger to AC
:TRIGger:COUPling?	Query returns AC

Edge Trigger

:TRIGger:EDGE:SLOPe

- **Command format**

:TRIGger:EDGE:SLOPe {POSitive|NEGative|ALTernation}

:TRIGger:EDGE:SLOPe?

- **Functional description**

Set the type of edge trigger to POSitive (rising edge), NEGative (falling edge) or ALTernation (rising/falling edge).

- **Return format**

  Query returns the type of edge trigger { POSitive | NEGative | ALTernation }.

- **For example**

  :TRIGger:EDGE:SLOP POS Set the edge trigger to POSitive

  :TRIGger:EDGE:SLOP? Query returns POSitive

Pulse Width Trigger

:TRIGger:PULSe:QUALifier

- **Command format**

  :TRIGger:PULSe:QUALifier {GREaterthan | LESSthan | INRange}

  :TRIGger:PULSe:QUALifier?

- **Functional description**

  Set the time condition for pulse, which is GREaterthan (greater than), LESSthan (less than), INRange (between).

- **Return format**

  Query returns {GREaterthan | LESSthan | INRange}.

- **For example**

  :TRIGger:PULSe:QUALifier GRE Set the pulse condition to GREaterthan

  :TRIGger:PULSe:QUALifier? Query returns GREaterthan

:TRIGger:PULSe:POLarity

- **Command format**

  :TRIGger:PULSe:POLarity {POSitive | NEGative}

  :TRIGger:PULSe:POLarity?

- **Functional description**

  Set the pulse polarity to POSitive (positive pulse width) or NEGative (negative pulse width).

- **Return format**

  Query returns { POSitive | NEGative }.

- **For example**

  :TRIGger:PULSe:POL POS Set the pulse polarity to POSitive

  :TRIGger:PULSe:POL? Query returns POSitive

:TRIGger:PULSe:TIME:UPPer

- **Command format**

:TRIGger:PULSe:TIME:UPPer <time>

:TRIGger:PULSe:TIME:UPPer?

- **Functional description**

Set the upper limit of time for pulse width trigger.

- **Return format**

Query returns the current time interval, the unit is s.

- **For example**

:TRIGger:PULSe:TIME:UPPer 1 Set the upper limit of pulse trigger to 1s

:TRIGger:PULSe:TIME:UPPer? Query returns 1.000e000

:TRIGger:PULSe:TIME:LOWer

- **Command format**

:TRIGger:PULSe:TIME:LOWer <time>

:TRIGger:PULSe:TIME:LOWer?

- **Functional description**

Set the lower limit of time for pulse width trigger.

- **Return format**

Query returns the current time interval, the unit is s.

- **For example**

:TRIGger:PULSe:TIME:LOWer 1 Set the lower limit of pulse trigger to 1s

:TRIGger:PULSe:TIME:LOWer? Query returns 1.000e000

Video Trigger

- **Command format**

:TRIGger:VIDeo:MODE { ODD | EVEN | LINE| ALINes}

:TRIGger:VIDeo:MODE?

- **Functional description**

Set the synchronous mode for video trigger, which includes ODD, EVEN, LINE and ALINes.

- **Return format**

Query returns { ODD | EVEN | LINE| ALIN}。

- **For example**

:TRIGger:VIDeo:MODE ODD

Set the synchronous mode of video trigger to ODD

:TRIGger:VIDeo:MODE? Query returns ODD

:TRIGger:VIDeo:STANdard

- **Command format**

  :TRIGger:VIDeo:STANdard { NTSC | PAL | SECAM }

  :TRIGger:VIDeo:STANdard?

- **Functional description**

  Set the video standard.

- **Return format**

  Query returns { NTSC | PAL | SECAM }.

- **For example**

  :TRIGger:VIDeo:STANdard NTSC Set the video standard to NTSC

  :TRIGger:VIDeo:STANdard? Query returns NTSC

:TRIGger:VIDEO:LINE

- **Command format**

  :TRIGger:VIDEO:LINE <value>

  :TRIGger:VIDEO:LINE?

- **Functional description**

  Set the specified line number for video synchronizing. <value> represents the specified line number, the range is related with the video standard.

- **Return format**

  Query returns the current specified line number.

- **For example**

  :TRIG:VIDEO:LINE 50

  Set the specified line number of video synchronizing to 50

  :TRIG:VIDEO:LINE? Query returns

Slope Trigger

:TRIGger:SLOPe:QUALifier

- **Command format**

  :TRIGger:SLOPe:QUALifier {GREaterthan | LESSthan | INRange}

  :TRIGger:SLOPe:QUALifier?

- **Functional description**

  Se the time condition for slope trigger, which is GREaterthan (greater than), LESSthan (less than), INRange (between).

- **Return format**

  Query returns {GREaterthan | LESSthan | INRange}.

- **For example**

  :TRIGger:SLOPe:QUALifier GRE Set the condition to GREaterthan

  :TRIGger:SLOPe:QUALifier? Query returns GREaterthan

:TRIGger:SLOPe:SLOPe

- **Command format**

  :TRIGger:SLOPe:SLOPe {POSitive|NEGative}

  :TRIGger:SLOPe:SLOPe?

- **Functional description**

  Set the type of slope trigger to POSitive (rising) or NEGative (falling).

- **Return format**

  Query returns {POSitive|NEGative}.

- **For example**

  :TRIGger:SLOPe:SLOPe POS Set the slope trigger to POSitive

  :TRIGger:SLOPe:SLOPe? Query returns POSitive

:TRIGger:SLOPe:TIME:UPPer

- **Command format**

  :TRIGger:SLOPe:TIME:UPPer <time>

  :TRIGger:SLOPe:TIME:UPPer?

- **Functional description**

  Set the upper limit of time for slope trigger.

- **Return format**

  Query returns the current time interval, the unit is s.

- **For example**

  :TRIGger:SLOPe:TIME:UPPer 1 Set the upper limit of slope trigger to 1s

  :TRIGger:SLOPe:TIME:UPPer? Query returns 1.000e000

:TRIGger:SLOPe:TIME:LOWer

- **Command format**

  :TRIGger:SLOPe:TIME:LOWer <time>

  :TRIGger:SLOPe:TIME:LOWer?

- **Functional description**

Set the lower limit of time for slope trigger.

- **Return format**

Query returns the current time interval, the unit is s.

- **For example**

:TRIGger:SLOPe:TIME:LOWer 1 Set the lower limit of slope trigger to 1s

:TRIGger:SLOPe:TIME:LOWer? Query returns 1.000e000

:TRIGger:SLOPe:THReshold

- **Command format**

:TRIGger:SLOPe:THReshold {LOW|HIGH|LH}

:TRIGger:SLOPe:THReshold?

- **Functional description**

Set the threshold mode for slop trigger.

- **Return format**

Query returns {LOW|HIGH|LH}.

- **For example**

:TRIGger:SLOPe:THR HIGH Set the threshold mode of slop trigger to HIGH

:TRIGger:SLOPe:THR? Query returns HIGH

:TRIGger:SLOPe:RATe:LOWer?

- **Command format**

:TRIGger:SLOPe:RATe:LOWer?

- **Functional description**

Query the currently lower limit of slop trigger, the unit is V/s.

- **Return format**

Query returns the lower limit of slop trigger by scientific notation.

- **For example**

:TRIGger:SLOPe:RATe:LOWer? Query returns 3.210E+000

:TRIGger:SLOPe:RATe:UPPer?

- **Command format**

:TRIGger:SLOPe:RATe:UPPer?

- **Functional description**

Query the currently upper limit of slop trigger, the unit is V/s.

- **Return format**

Query returns the upper limit of slop trigger by scientific notation.

- **For example**

:TRIGger:SLOPe:RATe:UPPer? Query returns 3.210E+000

Runt Trigger

:TRIGger:RUNT:QUALifier

- **Command format**

:TRIGger:RUNT:QUALifier {GREaterthan | LESSthan | INRange | NONE}

:TRIGger:RUNT:QUALifier?

- **Functional description**

Set time condition for runt level, which is GREaterthan (greater than), LESSthan (less than), INRange (between), NONE (arbitrary).

- **Return format**

Query returns {GREaterthan | LESSthan | INRange | NONE}.

- **For example**

:TRIGger:RUNT:QUALifier GRE Set the slope condition to GREaterthan

:TRIGger:RUNT:QUALifier? Query returns GREaterthan

:TRIGger:RUNT:POLarity

- **Command format**

:TRIGger:RUNT:POLarity {POSitive | NEGative}

:TRIGger:RUNT:POLarity?

- **Functional description**

Set the polarity of runt level to POSitive (positive pulse width) or NEGative (negative pulse width).

- **Return format**

Query returns {POSitive | NEGative}.

- **For example**

:TRIGger:RUNT:POL POS Set the polarity of runt level to POSitive

:TRIGger:RUNT:POL? Query returns POSitive

:TRIGger:RUNT:LEVel

- **Command format**

:TRIGger:RUNT:LEVel {LOW | HIGH | LH}

:TRIGger:RUNT:LEVel?

- **Functional description**

Set the trigger mode for runt level.

- **Return format**

Query returns {LOW | HIGH | LH}.

- **For example**

:TRIGger:RUNT:LEV HIGH	Set the runt level to HIGH
:TRIGger:RUNT:LEV?	Query returns HIGH

:TRIGger:RUNT:TIME:UPPer

- **Command format**

:TRIGger:RUNT:TIME:UPPer <time>

:TRIGger:RUNT:TIME:UPPer?

- **Functional description**

Set the upper limit of time for runt level.

- **Return format**

Query returns the currently upper limit of time, the unit is s.

- **For example**

:TRIGger:RUNT:TIME:UPPer 1	Set the upper limit of runt level to 1s
:TRIGger:RUNT:TIME:UPPer?	Query returns 1.000e000

:TRIGger:RUNT:TIME:LOWer

- **Command format**

:TRIGger:RUNT:TIME:LOWer <time>

:TRIGger:RUNT:TIME:LOWer?

- **Functional description**

Set the lower limit of time for runt level.

- **Return format**

Query returns the currently lower limit of time, the unit is s.

- **For example**

:TRIGger:RUNT:TIME:LOWer 1	Set the lower limit of runt level to 1s
:TRIGger:RUNT:TIME:LOWer?	Query returns 1.000e000

Over-amplitude Trigger

:TRIGger:WINDow:SLOPe

- **Command format**

  :TRIGger:WINDow:SLOPe {POSitive|NEGative|ALTernation}

  :TRIGger:WINDow:SLOPe?

- **Functional description**

  Set the trigger edge type to POSitive (rising edge), NEGative (falling edge) or ALTernation (rising/falling edge).

- **Return format**

  Query returns the edge type of trigger {POSitive|NEGative|ALTernation}.

- **For example**

  :TRIGger:WINDow:SLOP POS Set the window trigger to POSitive

  :TRIGger:WINDow:SLOP? Query returns POS

:TRIGger:WINDow:LEVel

- **Command format**

  :TRIGger:WINDow:LEVel {LOW | HIGH | LH}

  :TRIGger:WINDow:LEVel?

- **Functional description**

  Set the level mode for window trigger.

- **Return format**

  Query returns {LOW | HIGH | LH}.

- **For example**

  :TRIGger:WINDow:LEV HIGH Set the window trigger to HIGH

  :TRIGger:WINDow:LEV? Query returns HIGH

:TRIGger:WINDow:TIME

- **Command format**

  :TRIGger:WINDow:TIME <time>

  :TRIGger:WINDow:TIME?

- **Functional description**

  Set the time interval for window trigger.

- **Return format**

  Query returns the current time interval, the unit is s.

- **For example**

  :TRIGger:WINDow:TIME 1 Set the time interval of window trigger to 1s

  :TRIGger:WINDow:TIME? Query returns 1.000e000

:TRIGger:WINDow:POSition

- **Command format**

  :TRIGger:WINDow:POSition {ENTER|EXIT|TIME}

  :TRIGger:WINDow:POSition?

- **Functional description**

  Set the position of window trigger.

- **Return format**

  Query returns {ENTER|EXIT|TIME}.

- **For example**

  :TRIGger:WINDow:POS TIME Set the position of window trigger to TIME

  :TRIGger:WINDow:POS? Query returns TIME

Delay Trigger

:TRIGger:DELay:ARM:SOURce

- **Command format**

  :TRIGger:DELay:ARM:SOURce {CHANnel1| CHANnel2}

  :TRIGger:DELay:ARM:SOURce?

- **Functional description**

  Set the focus source for delay trigger.

- **Return format**

  Query returns {CHANnel1| CHANnel2}.

- **For example**

  :TRIGger:DELay:ARM:SOUR CHAN1 Set the focus source for CH1

  :TRIGger:DELay:ARM:SOUR? Query returns CHANnel1

:TRIGger:DELay:ARM:SLOPe

- **Command format**

  :TRIGger:DELay:ARM:SLOPe {NEGative | POSitive}

- :TRIGger:DELay:ARM:SLOPe?

- **Functional description**

Set the edge type of trigger source to POSitive (rising edge) or NEGative (falling edge).

- **Return format**

  Query returns {NEGative | POSitive}.

- **For example**

  :TRIGger:DELay:ARM:SLOPe NEG

  Set the edge type of trigger source to NEGative

  :TRIGger:DELay:ARM:SLOPe? Query returns NEGative

:TRIGger:DELay:TRIGger:SOURce

- **Command format**

  :TRIGger:DELay:TRIGger:SOURce {CHANnel1| CHANnel2}

  :TRIGger:DELay:TRIGger:SOURce?

- **Functional description**

  Set the trigger source for delay trigger.

- **Return format**

  Query returns {CHANnel1| CHANnel2}.

- **For example**

  :TRIGger:DELay:TRIGger:SOUR CHAN1 Set CH1 as the trigger source

  :TRIGger:DELay:TRIGger:SOUR? Query returns CHANnel1

:TRIGger:DELay:TRIGger:SLOPe

- **Command format**

  :TRIGger:DELay:TRIGger:SLOPe {NEGative | POSitive}

  :TRIGger:DELay:TRIGger:SLOPe?

- **Functional description**

  Set the edge type of trigger source to POSitive (rising edge) or NEGative (falling edge).

- **Return format**

  Query returns {NEGative | POSitive}.

- **For example**

  :TRIGger:DELay:TRIGger:SLOPe NEG

  Set the edge type of trigger source to NEGative

  :TRIGger:DELay:TRIGger:SLOPe? Query returns NEGative

:TRIGger:DELay:QUALifier

- **Command format**

:TRIGger:DELay:QUALifier { GREaterthan | LESSthan | INRange | OUTRange }

:TRIGger:DELay:QUALifier?

- **Functional description**

Set the time interval for delay trigger, which is GREaterthan (greater than), LESSthan (less than), INRange (beyond), OUTRange (within).

- **Return format**

Query returns { GREaterthan | LESSthan | INRange | OUTRange }.

- **For example**

:TRIGger:DELay:QUALifier GRE	Set the slope condition to GREaterthan
:TRIGger:DELay:QUALifier?	Query returns GREaterthan

:TRIGger:DELay:TIME:UPPer

- **Command format**

:TRIGger:DELay:TIME:UPPer <time>

:TRIGger:DELay:TIME:UPPer?

- **Functional description**

Set the upper limit of time for delay trigger.

- **Return format**

Query returns the currently upper limit of time, the unit is s.

- **For example**

:TRIGger:DELay:TIME:UPPer 1	Set the upper limit of delay trigger to 1s
:TRIGger:DELay:TIME:UPPer?	Query returns 1.000e000

:TRIGger:DELay:TIME:LOWer

- **Command format**

:TRIGger:DELay:TIME:LOWer <time>

:TRIGger:DELay:TIME:LOWer?

- **Functional description**

Set the lower limit of time for delay trigger.

- **Return format**

Query returns the currently lower limit of time, the unit is s.

- **For example**

:TRIGger:DELay:TIME:LOWer 1	Set the lower limit of delay trigger to 1s
:TRIGger:DELay:TIME:LOWer?	Query returns 1.000e000

:TRIGger:DELay:SELect

- **Command format**

  :TRIGger:DELay:SELect <SOURce<n>>

  :TRIGger:DELay:SELect

- **Functional description**

  Switching the selected source, SOURce<n> represents source, n take value as 1, 2. SOURce1 represents the focus source; SOURce2 represents the trigger source.

- **Return format**

  Query returns { SOURce1 | SOURce2 }.

- **For example**

  :TRIGger:DELay:SELect SOURce1 Select the focus source

  :TRIGger:DELay:SELect? Query returns SOURce1

Timeout Trigger

:TRIGger:TIMEOUT:TIME

- **Command format**

  :TRIGger:TIMEOUT:TIME <time>

  :TRIGger:TIMEOUT:TIME?

- **Functional description**

  Set the time interval for timeout trigger.

- **Return format**

  Query returns the current time interval, the unit is s.

- **For example**

  :TRIGger:TIMEOUT:TIME 1 Set the time interval of timeout trigger to 1s

  :TRIGger:TIMEOUT:TIME? Query returns 1.000e000

:TRIGger:TIMEOUT:SLOPe

- **Command format**

  :TRIGger:TIMEOUT:SLOPe {POSitive|NEGative|ALTernation}

  :TRIGger:TIMEOUT:SLOPe?

- **Functional description**

  Set the trigger edge to POSitive (rising edge), NEGative (falling edge) or ALTernation (rising/falling edge).

- **Return format**

Query returns the edge type of trigger {POSitive|NEGative|ALTernation}.

- **For example**

:TRIGger:TIMEOUT:SLOP POS Set the trigger edge to POSitive

:TRIGger:TIMEOUT:SLOP? Query returns POSitive

Duration Trigger

:TRIGger:DURation:PATTern

- **Command format**

:TRIGger:DURation:PATTern { HIGH | LOW | X }

:TRIGger:DURation:PATTern?

- **Functional description**

Set the code pattern of duration trigger to HIGH (code pattern is 1), LOW (code pattern is 0), X (the channel is invalid).

- **Return format**

Query returns { HIGH | LOW | X }.

- **For example**

:TRIGger:DURation:PATTern HIGH

Set the code pattern of duration trigger to 1

:TRIGger:DURation:PATTern? Query returns HIGH

:TRIGger:DURation:QUALifier

- **Command format**

:TRIGger:DURation:QUALifier { GREaterthan | LESSthan | INRange }

:TRIGger:DURation:QUALifier?

- **Functional description**

Set the time interval of delay trigger to GREaterthan (greater than), LESSthan (less than) or INRange (within).

- **Return format**

Query returns { GREaterthan | LESSthan | INRange }.

- **For example**

:TRIGger:DURation:QUALifier GRE

Set the slope condition to GREaterthan

:TRIGger:DURation:QUALifier? Query returns GREaterthan

:TRIGger:DURation:TIME:LOWer

- **Command format**

  :TRIGger:DURation:TIME:LOWer <time>

  :TRIGger:DURation:TIME:LOWer?

- **Functional description**

  Set the lower limit of time for duration trigger. When the timer interval is "greater than", the lower limit of time can be set.

- **Return format**

  Query returns the currently lower limit of time, the unit is s.

- **For example**

  :TRIGger:DURation:TIME:LOWer 1

  Set the lower limit of time for duration trigger to 1s

  :TRIGger:DURation:TIME:LOWer? Query returns 1.000e000

:TRIGger:DURation:TIME:UPPer

- **Command format**

  :TRIGger:DURation:TIME:UPPer <time>

  :TRIGger:DURation:TIME:UPPer?

- **Functional description**

  Set the upper limit of time for duration trigger. When the timer interval is "less than", the upper limit of time can be set.

- **Return format**

  Query returns the currently upper limit of time, the unit is s.

- **For example**

  :TRIGger:DURation:TIME:UPPer 1

  Set the upper limit of time for duration trigger to 1s

  :TRIGger:DURation:TIME:UPPer? Query returns 1.000e000

Setup & Hold Trigger

:TRIGger:SHOLd:DATA:SOURce

- **Command format**

  :TRIGger:SHOLd:DATA:SOURce {CHANnel1| CHANnel2}

  :TRIGger:SHOLd:DATA:SOURce?

- **Functional description**

Set the data source for setup & hold trigger.

- **Return format**

Query returns { CHANnel1| CHANnel2 }.

- **For example**

:TRIGger:SHOLd:DATA:SOUR CHAN1 Set CH1as the data source

:TRIGger:SHOLd:DATA:SOUR? Query returns CHANnel1

:TRIGger:SHOLd:CLOCK:SOURce

- **Command format**

:TRIGger:SHOLd:CLOCK:SOURce { CHANnel1| CHANnel2 }

:TRIGger:SHOLd:CLOCK:SOURce?

- **Functional description**

Set the clock source for setup & hold trigger.

- **Return format**

Query returns { CHANnel1| CHANnel2 }.

- **For example**

:TRIGger:SHOLd:CLOCK:SOUR CHAN1 Set CH1as the clock source

:TRIGger:SHOLd:CLOCK:SOUR? Query returns CHANnel1

TRIGger:SHOLd:SLOPe

- **Command format**

:TRIGger:SHOLd:SLOPe {POSitive|NEGative}

:TRIGger:SHOLd:SLOPe?

- **Functional description**

Set the trigger edge of setup & hold trigger to POSitive (rising edge) or NEGative (falling edge).

- **Return format**

Query returns {POSitive|NEGative}.

- **For example**

:TRIGger:SHOLd:SLOPe POS

Set the trigger edge of setup & hold trigger to POSitive

:TRIGger:SHOLd:SLOPe? Query returns POSitive

:TRIGger:SHOLd:PATTern

- **Command format**

:TRIGger:SHOLd:PATTern { HIGH | LOW }

:TRIGger:SHOLd:PATTern?

- **Functional description**

  Set the code pattern of setup & hold trigger to HIGH (code pattern is 1) or LOW (code pattern is 0).

- **Return format**

  Query returns { HIGH | LOW }.

- **For example**

  :TRIGger:SHOLd:PATTern HIGH

  Set the code pattern of setup & hold trigger to 1

  :TRIGger:SHOLd:PATTern? Query returns HIGH

:TRIGger:SHOLd:MODE

- **Command format**

  :TRIGger:SHOLd:MODE { SETup | HOLD }

  :TRIGger:SHOLd:MODE?

- **Functional description**

  Set time mode of setup & hold trigger to SETup (setup time) or HOLD (hold time).

- **Return format**

  Query returns { SETup | HOLD }.

- **For example**

  :TRIGger:SHOLd:MODE HOLD

  Set time mode of setup & hold trigger to HOLD

  :TRIGger:SHOLd:MODE? Query returns HOLD

:TRIGger:SHOLd:TIME

- **Command format**

  :TRIGger:SHOLd:TIME <time>

  :TRIGger:SHOLd:TIME?

- **Functional description**

  Set the time interval for setup & hold trigger.

- **Return format**

  Query returns the current time interval, the unit is s.

- **For example**

  :TRIGger:SHOLd:TIME 1

  Set the time interval of setup & hold trigger to 1s

:TRIGger:SHOLd:TIME? Query returns 1.000e000

:TRIGger:SHOLd:SELect

- **Command format**

  :TRIGger:SHOLd:SELect <SOURce<n>>

  :TRIGger:SHOLd:SELect

- **Functional description**

  Switching the selected source, SOURce<n> represents the source, n take value as 1, 2.

  SOURce1 represents the data source; SOURce2 represents the clock source.

- **Return format**

  Query returns { SOURce1 | SOURce2 }.

- **For example**

  :TRIGger:SHOLd:SELect SOURce1 Select the focus source

  :TRIGger:SHOLd:SELect? Query returns SOURce1

Nth Edge Trigger

:TRIGger:NEDGE:SLOPe

- **Command format**

  :TRIGger:NEDGE:SLOPe {POSitive|NEGative }

  :TRIGger:NEDGE:SLOPe?

- **Functional description**

  Set the trigger edge to POSitive (rising edge) or NEGative (falling edge).

- **Return format**

  Query returns the edge type of trigger {POSitive|NEGative }.

- **For example**

  :TRIGger:NEDGE:SLOP POS Set the trigger edge to POSitive

  :TRIGger:NEDGE:SLOP? Query returns POSitive

:TRIGger:NEDGE:TIME

- **Command format**

  :TRIGger:NEDGE:TIME <time>

  :TRIGger:NEDGE:TIME?

- **Functional description**

  Set the time interval of Nth edge trigger.

- **Return format**

  Query returns the current time interval, the unit is s.

- **For example**

  :TRIGger:NEDGE:TIME 1 Set the time interval of Nth edge trigger to 1s

  :TRIGger:NEDGE:TIME? Query returns 1.000e000

:TRIGger:NEDGE:VALue

- **Command format**

  :TRIGger:NEDGE:VALue <value>

  :TRIGger:NEDGE:VALue?

- **Functional description**

  Set the value of Nth edge, <value> is integer type and the range is 0~65535.

- **Return format**

  Query returns the current value of Nth edge.

- **For example**

  :TRIGger:NEDGE:VALue 100 Set the value of Nth edge to 100

  :TRIGger:NEDGE:VALue? Query returns 100

Code Pattern Trigger

:TRIGger:PATTern:PATTern

- **Command format**

  :TRIGger:PATTern:PATTern { HIGH | LOW | X | POSitive | NEGative }

  :TRIGger:PATTern:PATTern?

- **Functional description**

  Set the code pattern of trigger to HIGH (code pattern is 1), LOW (code pattern is 1), X (the channel is invalid), POSitive (rising) or NEGative (falling).

- **Return format**

  Query returns { HIGH | LOW | X | POSitive | NEGative }.

- **For example**

  :TRIGger:PATTern:PATTern HIGH Set the code pattern of trigger to 1

  :TRIGger:PATTern:PATTern? Query returns HIGH

CURSor Command

This command is used to set cursor parameter to measure the waveform data on the screen.

:CURSor:MODE

- **Command format**

  :CURSor:MODE { TRACK | INDependent }

  :CURSor:MODE?

- **Functional description**

  Set the cursor mode to TRACK or INDependent.

- **Return format**

  Query returns { TRACK | INDependent }.

- **For example**

:CURSor:MODE TRACK	Set the cursor mode to TRACK
:CURSor:MODE?	Query returns TRACK

:CURSor:TYPE

- **Command format**

  :CURSor:TYPE {AMPlitude | TIME | SCReen | CLOSe }

  :CURSor:TYPE?

- **Functional description**

  Set the cursor type of cursor mode to AMPlitude, TIME or CLOSe.

- **Return format**

  Query returns {AMPlitude | TIME | SCReen | CLOSe }.

- **For example**

:CURSor:TYPE AMP	Set the cursor type to AMPlitude
:CURSor:TYPE?	Query returns AMPlitude

:CURSor:SOURce

- **Command format**

  :CURSor:SOURce<source>

  :CURSor:SOURce?

- **Functional description**

  Set the cursor source of manual cursor mode.

  <source> take value as {CHANnel<n>|MATH | REFA | REFB }, n take value as 1, 2.

- **Return format**

Query returns {CHANnel<n>|MATH| REFA | REFB }.

- **For example**

:CURSor:SOURce CHAN1 Set CH1 as the cursor source

:CURSor:SOURce? Query returns CHANnel1

:CURSor:CURAX

- **Command format**

:CURSor:CURAX <value>

:CURSor:CURAX?

- **Functional description**

Set the horizontal position of A cursor line. The range of timebase YT mode is form left to right [0,699].

- **Return format**

Query returns the horizontal position of A cursor line.

- **For example**

:CURSor:CURAX 50

Set the horizontal position of A cursor line (manual) to 50

:CURSor:CURAX? Query returns 50

:CURSor:CURAY

- **Command format**

:CURSor:CURAY <value>

:CURSor:CURAY?

- **Functional description**

Set the vertical position of A cursor line. The range o is form up to down [28,227].

- **Return format**

Query returns the vertical position of A cursor line.

- **For example**

:CURSor:CURAY 50

Set the vertical position of A cursor line (manual) to 50

:CURSor:CURAY? Query returns 50

:CURSor:CURBX

- **Command format**

:CURSor:CURBX <value>

:CURSor:CURBX?

- **Functional description**

  Set the horizontal position of B cursor line. The range of timebase YT mode is form left to right [0,699].

- **Return format**

  Query returns the horizontal position of B cursor line.

- **For example**

  :CURSor:CURBX 50

  Set the horizontal position of B cursor line (manual) to 50

  :CURSor:CURBX? Query returns 50

:CURSor:CURBY

- **Command format**

  :CURSor:CURBY <value>

  :CURSor:CURBY?

- **Functional description**

  Set the vertical position of B cursor line. The range o is form up to down [28,227].

- **Return format**

  Query returns the vertical position of B cursor line.

- **For example**

  :CURSor:CURBY 50

  Set the vertical position of B cursor line (manual) to 50

  :CURSor:CURBY? Query returns 50

:CURSor:AXValue?

- **Command format**

  :CURSor:AXValue?

- **Functional description**

  Query X value at the cursor A, and the unit is determined by the amplitude unit of the currently selected channel.

- **Return format**

  Query returns X value at the cursor A by scientific notation.

- **For example**

  :CURSor:AXV? Query returns 2.000000E+02

:CURSor:AYValue?

- **Command format**

  :CURSor:AYValue?

- **Functional description**

  Query Y value at the cursor A, and the unit is determined by the amplitude unit of the currently selected channel.

- **Return format**

  Query returns Y value at the cursor A by scientific notation.

- **For example**

  :CURSor:AYV? Query returns 2.000000E+02

:CURSor:BXValue?

- **Command format**

  :CURSor:BXValue?

- **Functional description**

  Query X value at the cursor B, and the unit is determined by the amplitude unit of the currently selected channel.

- **Return format**

  Query returns X value at the cursor B by scientific notation.

- **For example**

  :CURSor:BXV? Query returns 2.000000E+02

:CURSor:BYValue?

- **Command format**

  :CURSor:BYValue?

- **Functional description**

  Query Y value at the cursor B, and the unit is determined by the amplitude unit of the currently selected channel.

- **Return format**

  Query returns Y value at the cursor B by scientific notation.

- **For example**

  :CURSor:BYV? Query returns 2.000000E+02

:CURSor:XDELta?

- **Command format**

:CURSor:XDELta?

- **Functional description**

  Query the difference △X between X at the cursor A and the cursor B in cursor track measurement.

- **Return format**

  Query returns the currently X difference between the cursor A and the cursor B by scientific notation.

- **For example**

  :CURSor:XDEL? Query returns 2.000000E+02

:CURSor:YDELta?

- **Command format**

  :CURSor:YDELta?

- **Functional description**

  Query the difference △Y between X at the cursor A and the cursor B in cursor track measurement. The unit is same as the unit of current channel.

- **Return format**

  Query returns the currently Y difference between the cursor A and the cursor B by scientific notation.

- **For example**

  :CURSor:YDEL? Query returns 2.000000E+02

:CURSor:XUNITs

- **Command format**

  :CURSor:XUNITs { SECond | HZ }

  :CURSor:XUNITs?

- **Functional description**

  Set the unit of horizontal cursor, SECond represents second, Hz represents Hertz.

- **Return format**

  Query returns { SECond | HZ }.

- **For example**

  :CURSor:XUNITs SECond Set the unit of horizontal cursor as SECond

  :CURSor:XUNITs? Query returns SECond

FILE Command

This command is used to set the reference waveform and storage function.

:FILE:LOAD

- **Command format**

  :FILE:LOAD <filename>[,<source>][,<disk>]

- **Functional description**

  Loading the waveform to the reference channel or the setup data.

  <filename> represents filename or file path, the filename must be character string with double quotation mark, such as "test.csv".

  - Filename with *.csv or *.wav indicates that the waveform data of a certain file is loaded to the reference channel, and it matched with the oscilloscope's suffix name.
  - Filename with *.dat indicates that the setup data of a certain file is loaded to the reference channel, and it matched with the oscilloscope's suffix name.

  <source > represents the reference channel { REFA | REFB }**, optional parameter, it only valid when loading the waveform data.**

  - REFA represents the reference channel A
  - REFB represents the reference channel B

  <disk> represents the storage medium { FLASH | UDISK **}, optional parameter. If it is omitted, it represents the internal data of FLASH.**

  - FLASH represents the internal data
  - UDISK represents the data in USB

- **For example**

  FILE:LOAD "test.wav", REFA, UISK

  Loading the waveform data "test.wav" from USB to the reference channel A

  FILE:LOAD "system-set-up01.dat"

  Loading the configuration data of 1 position from the internal medium to the oscilloscope

Note: when the oscilloscope cannot customize the filename and suffix,

- The filename of internal setup must be "system-set-up01.set"~ "system-set-up255.set", the maximum limit is 255 files.
- The filename of internal "bsv" file must be "wave01.bsv"~ " wave255.bsv", the maximum limit is 255 files.

:FILE:SAVE

- **Command format**

  :FILE:SAVE <filename>[,<source>][,<disk>][,<startframe>,<endframe>,<framerate>]

- **Functional description**

  Save the channel's waveform and setup data or recorded video to the file.

  <filename> represents the filename or file path, the filename must be character string with double quotation mark, such as "test.csv".

  - Save a channel's waveform of a certain file with the filename suffix of*.csv or *.wav to the file, and it matched with the oscilloscope's suffix name.
  - Save the setup data with the filename suffix of *.dat to the file, and it matched with the oscilloscope's suffix name.
  - Save a recorded video with the filename suffix of *.mp4 to the file, and it matched with the oscilloscope's suffix name.
  - Save a picture with the filename suffix of *.bmp, *.jpeg or*.png to the file, and it matched with the oscilloscope's suffix name.

  <source > represents the psychical channel { CHANnel1| CHANnel2 } **, optional parameter, it only valid when loading the waveform data.**

  - CHANnel1 represents the channel 1
  - CHANnel2 represents the channel 2

  <disk> represents the storage medium { FLASH | UDISK **}, optional parameter. If it is omitted, it represents the internal data of FLASH.**

  - FLASH represents the internal data
  - UDISK represents the data in USB

  <startframe> represents the start frame of recorded video.

  <endframe> represents the stop frame of recorded video.

  <framerate> represents the frame rate of recorded video.

- **For example**

  FILE:SAVE "test.csv",CHANnel1,UDISK

  The waveform data of the channel 1 saved as test.csv file to USB

  FILE:SAVE "system-set-up01.dat"

  The configuration data of oscilloscope saved as the internal medium 1 position

  FILE:SAVE "wave01.wav",CHANnel1,FLASH

  The waveform data of the channel 1 to the internal medium

  FILE:SAVE "wave01.wav",CHANnel1

The waveform data of the channel 1 to the internal medium

FILE:SAVE "system-set-up01.dat",FLASH

The configuration data of oscilloscope saved as the internal medium

FILE:SAVE "system-set-up01.dat"

The configuration data of oscilloscope saved as the internal medium 1 position

FILE:SAVE "test.mp4",1,100,10

The recorded video with the start frame 1, the stop frame 100 and the frame rate 10 saved as the file of test.mp4 to USB

FILE:SAVE "test.png",UDISK

The picture data with the format of png saved as the file of test.png to USB

Note: when the oscilloscope cannot customize the filename and suffix,

- The filename of internal setup must be "system-set-up01.set"~ "system-set-up255.set", the maximum limit is 255 files.
- The filename of internal "bsv" file must be "wave01.bsv"~ " wave255.bsv", the maximum limit is 255 files.

REFerence Command

This command is used to set the reference waveform function of the oscilloscope.

:REFerence:CLEar

- **Command format**

  :REFerence:CLEar {REFA|REFB|ALL}

- **Functional description**

  Clear the waveform of reference channel, ALL represents clear the waveform of all reference channel.

- **For example**

  :REFerence:CLEar REFA Clear the waveform of reference channel A

:REFerence:LOAD

- **Command format**

  :REFerence:LOAD <source>[,<ref>]

- **Functional description**

  Quickly loading the waveform of physical channel to the reference channel. If the reference channel is omitted, then load it to the currently selected reference channel.

  <source>:CHANnel<n>, n take value as 1, 2.

<ref>:{ REFA | REFB }, it represents the reference channel of A, B.

- **For example**

:REFerence:LOAD CHANnel1,REFB

Loading the waveform of CH1 to the reference channel

RECord Command

This command is used to set waveform recording function of the oscilloscope.

:RECord:ENABle

- **Command format**

:RECord:ENABle { {1|ON} | {0|OFF} }

:RECord:ENABle?

- **Functional description**

Turn on/off the waveform recording function.

- **Return format**

Query returns 1 or 0, it represents ON or OFF respectively.

- **For example**

:RECord:ENABle ON Turn on the waveform recording function

:RECord:ENABle?

Query returns 1, it represents that the waveform recording function is enabled

:RECord:STARt

- **Command format**

:RECord:STARt { {1|ON} | {0|OFF} }

:RECord:STARt?

- **Functional description**

Start (ON)/Stop (OFF) the waveform recording function.

- **Return format**

Query returns 1 or 0, it represents ON or OFF respectively.

- **For example**

:RECord:STARt ON The waveform recording is start

:RECord:STARt?

Query returns 1, it represents that the waveform recording function is start

:RECord:FAST

- **Command format**

  :RECord:FAST { {1|ON} | {0|OFF} }

  :RECord:FAST?

- **Functional description**

  Turn on/off the quick waveform recording function.

- **Return format**

  Query returns 1 or 0, it represents ON or OFF respectively.

- **For example**

  :RECord:FAST ON Turn on the quick waveform recording function

  :RECord:FAST?

  Query returns 1, it represents that the quick waveform recording function is turned on

:RECord:INTerval

- **Command format**

  :RECord:INTerval <time>

  :RECord:INTerval?

- **Functional description**

  Set the time interval for waveform recording.

- **Return format**

  Query returns the time interval for waveform recording by scientific notation and the unit is s.

- **For example**

  :RECord:INTerval 200ns Set the playback of waveform recording to 200 ns

  :RECord:INTerval? Query returns 2.000e-004

:RECord:PLAY

- **Command format**

  :RECord:PLAY { {1|ON} | {0|OFF} }

  :RECord:PLAY?

- **Functional description**

  Turn on/off the play of waveform recording.

- **Return format**

  Query returns 1 or 0, it represents ON or OFF respectively.

- **For example**

  :RECord:PLAY ON Play the waveform recording

:RECord:PLAY?

Query returns 1, it represents that start to play the waveform recording

:RECord:PLAY:DELay

- **Command format**

  :RECord:PLAY:DELay <time>

  :RECord:PLAY:DELay?

- **Functional description**

  Set the playback of waveform recording.

- **Return format**

  Query returns the playback of waveform recording by scientific notation and the unit is s.

- **For example**

  :RECord:PLAY:DELay 20ms Set the playback of waveform recording to 20 ms

  :RECord:PLAY:DELay? Query returns 2.000e-002

:RECord:CURRent

- **Command format**

  :RECord:CURRent <value>

  :RECord:CURRent?

- **Functional description**

  Set or query the current frame of waveform recording.

- **Return format**

  Query returns the current frame of waveform recording, it is integer type.

- **For example**

  :RECord:CURRent 100 Set the current frame as 100

  :RECord:CURRent? Query returns 100

:RECord:FRAMes

- **Command format**

  :RECord:FRAMes <value>

  :RECord:FRAMes?

- **Functional description**

  Set or query the frame number of waveform recording.

- **Return format**

  Query returns the frame number of waveform recording, it is integer type.

- **For example**

:RECord:FRAMes 400	Set the frame number as 400
:RECord:FRAMes?	Query returns 400

DVM Command

This command is used to set the function of digital voltmeter.

:DVM:ENABle

- **Command format**

  :DVM:ENABle { {1|ON} | {0|OFF} }

  :DVM:ENABle?

- **Functional description**

  Set or query the function (ON/OFF) of digital voltmeter.

- **Return format**

  Query returns 1 or 0, it represents ON or OFF respectively.

- **For example**

:DVM:ENABle ON	Turn on the digital voltmeter
:DVM:ENABle?	Query returns 1

:DVM:SOURCe

- **Command format**

  :DVM:SOURCe <source>

  :DVM:SOURCe?

- **Functional description**

  Set or query the source of digital voltmeter.

  <source>:CHANnel<n>, n take value as 1, 2.

- **Return format**

  Query returns { CHANnel1 | CHANnel2 }.

- **For example**

:DVM:SOURCe CHANnel1	Set CH1 as the source
:DVM:SOURCe?	Query returns CHANnel1

:DVM:MODE

- **Command format**

  :DVM:MODE {ACRMs|DC|DCRMs}

  :DVM:MODE?

- **Functional description**

  Set or query the mode of digital voltmeter.

- **Return format**

  Query returns {ACRMs|DC|DCRMs}.

- **For example**

  :DVM:MODE DC Set the mode of digital voltmeter to DC

  :DVM:MODE? Query returns DC

:DVM:CURRent?

- **Command format**

  : DVM:CURRent?

- **Functional description**

  Query the currently measured voltage of digital voltmeter.

- **Return format**

  Query returns the currently measured voltage of digital voltmeter by scientific notation and the unit is determined by the command :DVM:MODE.

- **For example**

  :DVM:CURRent? The currently measured voltage is 3.000e-003

:DVM:LIMit:ENABle

- **Command format**

  :DVM:LIMit:ENABle { {1|ON} | {0|OFF} }

  :DVM:LIMit:ENABle?

- **Functional description**

  Turn on/off the alarm function of digital voltmeter.

- **Return format**

  Query returns 1 or 0, it represents ON or OFF respectively.

- **For example**

  :DVM:LIMit:ENABle ON Turn on the alarm function of digital voltmeter

  :DVM:LIMit:ENABle? Query returns 1

:DVM:LIMit:QUALifier

- **Command format**

  :DVM:LIMit:QUALifier {GREaterthan | LESSthan | INRange | OUTRange}

  :DVM:LIMit:QUALifier?

- **Functional description**

  Set the alarm condition for digital voltmeter, which is GREaterthan (greater than), LESSthan (less than), INRange (between), OUTRange (beyond).

- **Return format**

  Query returns { GREaterthan | LESSthan | INRange | OUTRange }.

- **For example**

:DVM:LIMit:QUALifier GRE	Set the alarm condition to GREaterthan
:DVM:LIMit:QUALifierr?	Query returns GREaterthan

:DVM:LIMit:UPPer

- **Command format**

  :DVM:LIMit:UPPer <upper>

  :DVM:LIMit:UPPer?

- **Functional description**

  Set the upper limit of alarm for digital voltmeter.

- **Return format**

  Query returns the currently upper limit by scientific notation and the unit is V.

- **For example**

:DVM:LIMit:UPPer 2V	Set the upper limit to 2 V
:DVM:LIMit:UPPer?	Query returns 2.000e000

:DVM:LIMit:LOWer

- **Command format**

  :DVM:LIMit:LOWer <lower >

  :DVM:LIMit:LOWer?

- **Functional description**

  Set the lower limit of alarm for digital voltmeter.

- **Return format**

  Query returns the currently lower limit by scientific notation and the unit is V.

- **For example**

:DVM:LIMit:LOWer 1V	Set the lower limit to 1 V
:DVM:LIMit:LOWer?	Query returns 1.000e000

PF Command

This command is used to set Pass/Fail test function.

:PF:ENABle

- **Command format**

  :PF:ENABle { {1|ON} | {0|OFF} }

  :PF:ENABle?

- **Functional description**

  Set or query the function (ON/OFF) of Pass/Fail test.

- **Return format**

  Query returns 1 or 0, it represents ON or OFF respectively.

- **For example**

:PF:ENABle ON	Turn on the function of Pass/Fail test
:PF:ENABle?	Query returns 1

:PF:SOURCe

- **Command format**

  :PF:SOURCe <source>

  :PF:SOURCe?

- **Functional description**

  Set or query the measuring source of Pass/Fail test.

  <source>:CHANnel<n>, n take value as 1, 2.

- **Return format**

  Query returns { CHANnel1 | CHANnel2 }.

- **For example**

:PF:SOURCe CHANnel1	Set CH1 as the measuring source
:PF:SOURCe?	Query returns CHANnel1

:PF:OPERate

- **Command format**

  :PF:OPERate {RUN|STOP}

  :PF:OPERate?

- **Functional description**

  Run/stop the Pass/Fail test.

- **Return format**

Query returns {RUN|STOP}.

- **For example**

:PF:OPERate RUN	Run the Pass/Fail test
:PF:OPERate?	Query returns RUN

:PF:OUTPut

- **Command format**

:PF:OUTPut {PASS|FAILED}

:PF:OUTPut?

- **Functional description**

Set or query the output of Pass/Fail test.

- **Return format**

Query returns {PASS|FAILED}.

- **For example**

:PF:OUTPut PASS	Set the output of Pass/Fail test to PASS
:PF:OUTPut?	Query returns PASS

:PF:STOP:TYPe

- **Command format**

:PF:STOP:TYPe {PCOUNT|FCOUNT}

:PF:STOP:TYPe?

- **Functional description**

Set or query the stop type of Pass/Fail test.

PCOUNT represents the count of pass; FCOUNT represents the count of fail.

- **Return format**

Query returns {PCOUNT|FCOUNT}.

- **For example**

:PF:STOP:TYPe PCOUNT	Set the stop type of Pass/Fail test as PCOUNT
:PF:STOP:TYPe?	Query returns PCOUNT

:PF:STOP:QUALifier

- **Command format**

:PF:STOP:QUALifier {LEQual | GEQual}

:PF:STOP:QUALifier?

- **Functional description**

Set or query the stop condition of Pass/Fail test.

GEQual represents greater than or equal to; LEQual represents less than or equal to.

- **Return format**

Query returns {LEQual | GEQual}.

- **For example**

:PF:STOP:QUALifier GEQual

Set the stop condition of Pass/Fail test as ≥ (GEQual)

:PF:STOP:QUALifier? Query returns GEQual

:PF:STOP:THReshold

- **Command format**

:PF:STOP:THReshold <value>

:PF:STOP:THReshold?

- **Functional description**

Set or query the stop threshold of Pass/Fail test.

<value>: the stop threshold, the range is 1~30000, the specified range will self-adjusting according to the oscilloscope.

- **Return format**

Query returns the stop threshold, it is integer data.

- **For example**

:PF:STOP:THReshold 100 Set the stop threshold of Pass/Fail test as 100

:PF:STOP:THReshold? Query returns 100

:PF:TEMPlate:SOURce

- **Command format**

:PF:TEMPlate:SOURce <source>

:PF:TEMPlate:SOURce?

- **Functional description**

Set or query the reference channel that set by the template of Pass/Fail test.

<source>: CHANnel<n>, n take value as 1, 2.

- **Return format**

Query returns { CHANnel1 | CHANnel2 }.

- **For example**

:PF:TEMPlate:SOURce CHANnel1

Set the reference channel of the template as CH1

:PF:TEMPlate:SOURce? Query returns CHANnel1

:PF:TEMPlate:X

- **Command format**

  :PF:TEMPlate:X <value>

  :PF:TEMPlate:X?

- **Functional description**

  Set or query the horizontal tolerance of the template of Pass/Fail test.

  <value>: the horizontal tolerance, the range is 1~100, the specified range will self-adjusting according to the oscilloscope.

- **Return format**

  Query returns the horizontal tolerance of the template, it is integer data.

- **For example**

  :PF:TEMPlate:X 50

  Set the horizontal tolerance of the template as 50

  :PF:TEMPlate:X? Query returns 50

:PF:TEMPlate:Y

- **Command format**

  :PF:TEMPlate:Y <value>

  :PF:TEMPlate:Y?

- **Functional description**

  Set or query the vertical tolerance of the template of Pass/Fail test.

  <value>: the vertical tolerance, the range is 1~100, the specified range will self-adjusting according to the oscilloscope.

- **Return format**

  Query returns the vertical tolerance of the template, it is integer data.

  For example

  :PF:TEMPlate:Y 50

  Set the vertical tolerance of the template as 50

  :PF:TEMPlate:Y? Query returns 50

:PF:RESult?

- **Command format**

  :PF:RESult?

- **Functional description**

  Set or query the statistic results of Pass/Fail test.

  Format of returned data: <pass>,<failed>,<total> , <pass> represents the count of pass , <failed> represents the count of fail, <total> represents the total count.

- **Return format**

  Query returns the statistic results of Pass/Fail test.

- **For example**

  :PF:RESult? Query returns 35, 42, 77

ACQuire Command

This command is used to set the sampling mode of the oscilloscope.

:ACQuire:TYPE

- **Command format**

  :ACQuire:TYPE {NORMal | AVERage | PEAKdetect | HRESolution }

  :ACQuire:TYPE?

- **Functional description**

  Set the sampling mode of the oscilloscope, including NORMal (normal), AVERage (average), PEAKdetect (peak) and HRESolution (high resolution).

- **Return format**

  Query returns {NORMal | AVERage | PEAKdetect | HRESolution }.

- **For example**

  :ACQ:TYPE AVER Set the sampling mode to AVER

  :ACQ:TYPE? Query returns AVERage

:ACQuire:AVERages:COUNt

- **Command format**

  :ACQuire:AVERages:COUNt <count>

  :ACQuire:AVERages:COUNt?

- **Functional description**

  Set the count of average sampling mode, <count> stepped as Nth power of 2, take value form 2~8192, 1≤N≤30.

- **Return format**

  Query returns the current count of average sampling.

- **For example**

:ACQ:AVER:COUN 32	Set the count of average sampling to 32
:ACQ:AVER:COUN?	Query returns 32

:ACQuire:MEMory:DEPTh

- **Command format**

  :ACQuire:MEMory:DEPTh { AUTO | 7K | 70K | 700K | 7M | 28M | 56M }

  :ACQuire:MEMory:DEPTh?

- **Functional description**

  Set the storage depth, it will self-adjusting for different model.

- **Return format**

  Query returns { AUTO | 7K | 70K | 700K | 7M | 28M | 56M }.

- **For example**

:ACQ:MEM:DEPT AUTO	Se the storage depth to AUTO
:ACQ:MEM:DEPT?	Query returns AUTO

DISPlay Command

This command is used to set or query the display function or data of the oscilloscope.

:DISPlay:DATA:FORMat

- **Command format**

  :DISPlay:DATA:FORMat { BMP | JPEG | PNG}

  :DISPlay:DATA:FORMat?

- **Functional description**

  Set the data format of image and local saved image.

- **Return format**

  Query returns the image format { BMP | JPEG | PNG}.

- **For example**

:DISPlay:DATA:FORMat PNG	Set the image data of PNG format
:DISPlay:DATA:FORMat?	Query returns PNG

:DISPlay:DATA?

- **Command format**

  :DISPlay:DATA?

- **Functional description**

Query the image data of oscilloscope on the screen, it returns the image data in format of BMP by default. The format of returned image data is determined by the command :DISPlay:DATA:FORMat.

- **Return format**

  Query returns the image data, the returned data is conform with Appendix 2:IEEE 488.2 Binary System Data Format.

- **For example**

  :DISPlay:DATA? Query returns the image data

:DISPlay:FORMat

- **Command format**

  :DISPlay:FORMat { VECTors | DOTS }

  :DISPlay:FORMat?

- **Functional description**

  Set the display format for sampling point, including VECTors (vector display) and DOTS (directly display).

- **Return format**

  Query returns { VECTors | DOTS }.

- **For example**

  :DISPlay:FORMat VECT Set the display format to VECT

  :DISPlay:FORMat? Query returns VECTors

:DISPlay:GRID

- **Command format**

  :DISPlay:GRID {FULL|HALF|CROSS|NONE}

  :DISPlay:GRID?

- **Functional description**

  Set the display mode for grid on the screen.

- **Return format**

  Query returns {FULL|HALF|CROSS|NONE}.

- **For example**

  :DISPlay:GRID CROSS

  Set to display cross coordinate line and not display grid

  :DISPlay:GRID? Query returns CROSS

:DISPlay:GRID:BRIGhtness

- **Command format**

  :DISPlay:GRID:BRIGhtness <count>

  :DISPlay:GRID:BRIGhtness?

- **Functional description**

  Set the grid brightness, <count> tale value as 1~100, the larger the number, the brighter the grid.

- **Return format**

  Query returns the current grid brightness.

- **For example**

  :DISPlay:GRID:BRIGhtness 50 Set the grid brightness to 50

  :DISPlay:GRID:BRIGhtness? Query returns 50

:DISPlay:GRAD:TIME

- **Command format**

  :DISPlay:GRAD:TIME {DSO|MINimum|50ms|100ms|200ms|500ms|1s|2s|5s|10s|20s|INFinite}

  :DISPlay:GRAD:TIME?

- **Functional description**

  Set the persistence, DSO represents that the persistence is closed.

- **Return format**

  Query returns

  {DSO|MINimum|50ms|100ms|200ms|500ms|1s|2s|5s|10s|20s|INFinite}.

- **For example**

  :DISPlay:GRAD:TIME 50ms Set the persistence to 50 ms

  :DISPlay:GRAD:TIME? Query returns 50 ms

:DISPlay:COLOR

- **Command format**

  :DISPlay:COLOR { {1|ON} | {0|OFF} }

  :DISPlay:COLOR?

- **Functional description**

  Turn on/off the color temperature.

- **Return format**

  Query returns 1 or 0, it represents ON or OFF respectively.

- **For example**

:DISPlay:COLOR ON Turn on the color temperature

:DISPlay:COLOR?

Query returns 1, it represents that the color temperature is displayed

:DISPlay:COLOR:INVERt

- **Command format**

:DISPlay:COLOR:INVERt { {1|ON} | {0|OFF} }

:DISPlay:COLOR:INVERt?

- **Functional description**

Turn on/off the reversed color.

- **Return format**

Query returns 1 or 0, it represents ON or OFF respectively.

- **For example**

:DISPlay:COLOR:INVERt ON Turn on the reversed color

:DISPlay:COLOR:INVERt?

Query returns 1, it represents that the reversed color is displayed

:DISPlay:WAVE:BRIGhtness

- **Command format**

:DISPlay:WAVE:BRIGhtness <count>

:DISPlay:WAVE:BRIGhtness?

- **Functional description**

Set the waveform brightness, <count> take value as 1~100, the bigger the number, the brighter the waveform.

- **Return format**

Query returns the current waveform brightness.

- **For example**

:DISPlay:WAVE:BRIGhtness 50 Set the waveform brightness to 50

:DISPlay:WAVE:BRIGhtness? Query returns 50

:DISPlay:TRANsparent

- **Command format**

:DISPlay:TRANsparent { {1|ON} | {0|OFF} }

:DISPlay:TRANsparent?

- **Functional description**

Turn on/off the display transparence.

- **Return format**

  Query returns 1 or 0, it represents ON or OFF respectively.

- **For example**

  :DISPlay:TRANsparent ON Turn on the display transparence

  :DISPlay:TRANsparent?

  Query returns 1, it represents that the display transparence is turned on.

:DISPlay:CLEar

- **Command format**

  :DISPlay:CLEar

- **Functional description**

  Clear and refresh the waveform of oscilloscope on the screen. If there is the reference waveform, clearing and refreshing the reference waveform.

:DISPlay:TYPE

- **Command format**

  :DISPlay:TYPE {XY |YT}

  :DISPlay:TYPE?

- **Functional description**

  Set the display mode of timebase as XY (X-Y mode: display the amplitude of channel 1 on the horizontal axis, display the amplitude of channel 2 on the vertical axis); YT （Y-T mode: display the relative relationship of vertical voltage and horizontal time）.

- **Return format**

  Query returns {XY|YT}.

- **For example**

  :DISP:TYPE YT Set the display mode of timebase as YT

  :DISP:TYPE? Query returns YT

WAVeform Command

This command is used to read the waveform data and relative parameter on the screen of the oscilloscope.

:WAVeform:MODE

- **Command format**

  :WAVeform:MODE {NORMal | RAW}

  :WAVeform:MODE?

- **Functional description**

  NORMal: read the current waveform data on the screen, the count of waveform data is fixed count

  RAW: read the waveform data from internal storage, the count of waveform data is related to storage depth.

  Note: This instruction resets the count of start, cut-off and waveform. Data in internal storage can only be read when the oscilloscope in stop status. This instruction is invalid in MATH channel.

- **Return format**

  Query returns {NORMal | RAW}.

- **For example**

  :WAVeform:MODE RAW Set the read mode of waveform data to RAW

  :WAVeform:MODE? Query returns RAW

:WAVeform:FORMat

- **Command format**

  :WAVeform:FORMat { WORD | BYTE | ASCII }

  :WAVeform:FORMat?

- **Functional description**

  The oscilloscope sets the format of waveform data to AD by default.

  BYTE: return AD data, which is a waveform point take one byte (8 bits)

  WORD: return AD data, which is a waveform point take two bytes (16 bits), low 8-bit is valid, high 8-bit is 0.

  ASCII: the returned waveform is return the actual voltage of each waveform points by scientific notation, each voltage is separate by comma and conform with

  Appendix 2:IEEE 488.2 Binary System Data Format.

  For example, #412342.00000E+01, 2.20000E+01, 2.30000E+01.....\n.

- **Return format**

  Query returns { WORD | BYTE | ASCII }.

- **For example**

  :WAVeform:FORMat BYTE Return format of AD data is single byte mode

:WAVeform:FORMat? Query returns BYT

:WAVeform:STARt

- **Command format**

  :WAVeform:STARt <start>

  :WAVeform:STARt?

- **Functional description**

  Set or query the start point of waveform data reading, <start> is integer data.

  NORMal: 1~1400

  RAW: 1 ~ the currently maximum point of storage depth

- **Return format**

  Query returns the start point.

- **For example**

  :WAVeform:STARt 200 Set the start point of waveform data reading to 200

  :WAVeform:STARt? Query returns 200

:WAVeform:STOP

- **Command format**

  :WAVeform:STOP <stop>

  :WAVeform:STOP?

- **Functional description**

  Set or query the cut-off point of waveform data reading, <stop> is integer data.

  NORMal: < stop> range is from 1 to 1400

  RAW: < stop> range is from 1 to the currently maximum point of storage depth

- **Return format**

  Query returns the cut-off point.

- **For example**

  :WAVeform:STOP 400 Set the cut-off point of waveform data reading to 400

  :WAVeform:STOP? Query returns 400

:WAVeform:SOURce

- **Command format**

  :WAVeform:SOURce {CHANnel<n>| MATH}

  :WAVeform:SOURce?

- **Functional description**

Set the signal source of waveform data to be queried. If the command is not sent, it represents that query the waveform data of the current channel.

- **Return format**

  Query returns { CHANnel1 | CHANnel2| MATH }.

- **For example**

  :WAVeform:SOURce CHAN1

  Set CH1 as the signal source of waveform data to be queried

  :WAVeform:SOURce? Query returns CHANnel1

:WAVeform:POINts

- **Command format**

  :WAVeform:POINts <points>

  :WAVeform:POINts?

- **Functional description**

  Set the waveform point to be returned, the default value is 0.

- **Return format**

  Query returns the waveform point to be returned.

- **For example**

  :WAVeform:POINts 120 Set the waveform point to 120

  :WAVeform:POINts? Query returns 120

:WAVeform:DATA?

- **Command format**

  :WAVeform:DATA?

- **Functional description**

  Read the waveform data in the specified data source.

- **Return format**

  WAVeform:POINts is the specified quantity of the waveform data, the waveform data source is related with the command:WAVeform:SOURce, the data format is related with the command WAVeform:FORMat, returned data is conform with Appendix 2:IEEE 488.2 Binary System Data Format.

- **For example**

  The instruction executing order to obtain the waveform data of the specified data source in the following order.

  - Acquire flow of waveform data on the screen

:WAVeform:SOURce CHAN1

Set CH1 as the signal source of waveform data to be queried

:WAVeform:MODE NORMal

Read the waveform data on the screen

:WAVeform:FORMat BYTE

Set the return format of waveform data to AD single byte mode

:WAVeform:DATA?

Acquire the waveform data

◆ Acquire flow of the waveform data in RAM, this flow only valid in the stop state

:WAVeform:SOURce CHAN1

Set CH1 as the signal source of waveform data to be queried

:WAVeform:MODE RAW

Read the waveform data in RAM

:WAVeform:FORMat BYTE

Set the return format of waveform data to AD single byte mode

:WAVeform:POINts 5000 The waveform point in RAM is 5000

Cycle reading the waveform point in RAM

{

:WAVeform:DATA? Acquire a batch of waveform data in RAM

:WAVeform:START? The start point of waveform data reading，-1 represents that is the last point

}

Explanation: when reading internal data in batch, the read back data of each time is only the data of an area in the memory, and the waveform data between two adjacent pieces is continuous. Each batch data is conform with Appendix 2:IEEE 488.2 Binary System Data Format.

:WAVeform:PREamble?

- **Command format**

:WAVeform:PREamble?

- **Functional description**

Query returns the waveform setup parameter of the current system.

- **Return format**

Query returns the parameter that separate by comma “,”.

Return data foramt: Format,Type,Points,Count,Xinc,Xor,Xref,Yinc,Yor,Yref.

Format: BYTE (0), WORD (1), ASCII(2)

Type: NORMAL(0), PEAK(1), AVER(2), HRESolution(3)

Points: the waveform data point to be return

Count: it is the average count in the average sampling, it is 1 in other mode

Xinc: the difference time between two points of waveform data source in X direction

Xor: the relative time of trigger point

Xref: X reference

Yinc: unit of voltage in Y direction

Yor: Y direction with respect to the zero position of YREF

Yref: reference of Y direction, the channel zero level ADC

- **For example**

:WAVeform:PREamble? is return

1,0,0,1,8.000e-009,-6.000e-006,0,4.000e-002,10,128

:WAVeform:XINCrement?

- **Command format**

:WAVeform:XINCrement?

- **Functional description**

Query time interval between two adjacent points of the currently selected channel source in X direction.

Returned value is related to the current data reading mode.

In NORMal mode，XINCrement=TimeScale/waveform point of time scale in horizontal poistion (50)

In RAW mode，XINCrement=1/SampleRate

- **Return format**

Query returns the timebase, the unit is s.

- **For example**

:WAV:XINC? Query returns 3.000e-003

:WAVeform:XORigin?

- **Command format**

:WAVeform:XORigin?

- **Functional description**

Query the start time of waveform data of the currently selected channel source in X direction.

Returned value is related to the current data reading mode, time in trigger point is zero, before

the trigger point is negative.

In NORMal mode, return the start time of waveform data displayed on the screen: XORigin = -1*TimeScale*7

In RAW mode, return the start time of waveform data in RAW, XORigin = -1* (SamplePoints/SampleRate)/2

- **Return format**

Query returns the time, the unit is s.

- **For example**

:WAV:XOR? Query returns 3.000e-002

:WAVeform:XREFerence?

- **Command format**

:WAVeform:XREFerence?

- **Functional description**

Query the reference time benchmark of waveform point of the currently selected channel source in X direction, it's always zero.

- **Return format**

Query the reference time benchmark, it returns 0.

- **For example**

:WAV:XREF? Query returns 0

:WAVeform:YINCrement?

- **Command format**

:WAVeform:YINCrement?

- **Functional description**

Query the unit of voltage of current selected channel source in Y direction, the unit is the same with the current amplitude unit.

Returned value is related to the current data reading mode: YINCrement = VerticalScale/waveform point of amplitude scale in vertical position (25)

- **Return format**

Query returns the unit of voltage in Y direction.

- **For example**

:WAV:YINC? Query returns 2.000e000

:WAVeform:YORigin?

- **Command format**

  :WAVeform:YORigin?

- **Functional description**

  Query the vertical offset in the Y direction of the currently selected channel source with respect to the vertical reference position.

  Returned value is related to the current data reading mode, up is positive and down is negative based on the reference: YORigin = VerticalOffset/YINCrement

- **Return format**

  Query returns the vertical offset, it is integer type.

- **For example**

  :WAV:YOR? Query returns 10

:WAVeform:YREFerence?

- **Command format**

  :WAVeform:YREFerence?

- **Functional description**

  Query the vertical reference position of the current selected channel source in Y direction, and ADC value of channel zero level.

  Returned value is related to the current data reading mode: YREFerence is fixed as 128

- **Return format**

  Query returns the reference position, it is integer type.

- **For example**

  :WAV:YREF? Query returns 128

SBUS Command

This command is used to set bus decoding and trigger parameter of RS232, SPI and I^2C.

Basic Attribute

:SBUS:DISPlay

- **Command format**

  :SBUS:DISPlay { {1|ON} | {0|OFF} }

  :SBUS:DISPlay?

- **Functional description**

  Turn on/off the bus decoding of the oscilloscope.

- **Return format**

  Query returns 1 or 0, it represents ON or OFF respectively.

- **For example**

  :SBUS:DISPlay ON

  Turn on the bus decoding, displaying the decoded waveform

  :SBUS:DISPlay? Query returns 1

:SBUS:MODE

- **Command format**

  :SBUS:MODE { RS232 | I2C | SPI }

  :SBUS:MODE?

- **Functional description**

  Select the bus decoding and trigger mode for the oscilloscope.

- **Return format**

  Query returns { RS232 | I2C | SPI }.

- **For example**

  :SBUS:MODE I2C Select I^2C as the bus decoding

  :SBUS:MODE? Query returns I2C

:SBUS:BASE

- **Command format**

  :SBUS:BASE {ASCII | BINary | HEX | DEC}

  :SBUS:BASE?

- **Functional description**

  Set the display format for bus decoding.

- **Return format**

  Query returns {ASCII | BINary | HEX | DEC}.

- **For example**

  :SBUS:BASE BIN Set the bus decoding to binary display format

  :SBUS:BASE? Query returns BINary

:SBUS:EVENt

- **Command format**

:SBUS:EVENt { {1|ON} | {0|OFF} }

:SBUS:EVENt?

- **Functional description**

Turn on/off the bus event of the oscilloscope.

- **Return format**

Query returns 1 or 0, it represents ON or OFF respectively.

- **For example**

:SBUS:EVENt ON	Turn on the bus event
:SBUS:EVENt?	Query returns 1

:SBUS:DATA?

- **Command format**

:SBUS:DATA?

- **Functional description**

Read the data of decoding event list.

- **Return format**

Query returns the data in decoding event list, returned data is conform with Appendix 2:IEEE 488.2 Binary System Data Format.

- **For example**

:SBUS:DATA? Query returns :

#9000000089RS232,

TIME,DATA,CHECK,

-1us,0,0,

-890.5ns,1,0,

-403.4ns,0,0,

9.8ns,1,0,

531.7ns,0,0,

RS232 represents decoding type (it may be I^2C or SPI), the event table data in CSV format followed behind. The specified format of the event table data is automatically adapted by different devices. The data are separated by commas and will automatically line wrap according to the decoding list, the data value is related to the setting system display.

:SBUS:TRIGger:MODE

- **Command format**

:SBUS:TRIGger:MODE {RS232 | I2C | SPI }

:SBUS:TRIGger:MODE?

- **Functional description**

Set the bus trigger mode of the oscilloscope.

- **Return format**

Query returns {RS232 | I2C | SPI }.

- **For example**

:SBUS:TRIGger:MODE I2C	Set to I^2C
:SBUS:TRIGger:MODE?	Query returns I^2C

:SBUS:VERTical:POSition

- **Command format**

:SBUS:VERTical:POSition <value>

:SBUS:VERTical:POSition?

- **Functional description**

Set the trigger vertical position of the bus trigger. The parameter is integer, step is 6, range is [-160,160]. The center of screen is zero point, up is positive, down is negative.

- **Return format**

Query returns the vertical position.

- **For example**

:SBUS:VERTical:POSition 10	The vertical position value of bus is 10
:SBUS:VERTical:POSition?	Query returns 10

:SBUS:TRIGger:SWEep

- **Command format**

:SBUS:TRIGger:SWEep {AUTO|NORMal|SINGle}

:SBUS:TRIGger:SWEep?

- **Functional description**

Select the sweep mode for bus trigger.

AUTO: in no trigger condition, the internal will produce a trigger signal to force generate.

NORMal: it will be generated only the trigger condition is met.

SINGle: it will be generated one time and stop when trigger condition is met.

- **Return format**

Query returns the trigger sweep mode { AUTO|NORMal|SINGle }.

- **For example**

:SBUS:TRIGger:SWEep AUTO	Set the bus trigger sweep mode to AUTO

:SBUS:TRIGger:SWEep? Query returns AUTO

RS232

:SBUS:RS232:BAUDrate

- **Command format**

  :SBUS:RS232:BAUDrate <baudrate>

  :SBUS:RS232:BAUDrate?

- **Functional description**

  Set RS232 decoding baud rate of the oscilloscope. The parameter is integer, and the range is 120~5000000.

- **Return format**

  Query returns the baud rate.

- **For example**

  :SBUS:RS232:BAUDrate 500 Set RS232 baud rate to 500 b/s

  :SBUS:RS232:BAUDrate? Query returns 500

:SBUS:RS232:BITorder

- **Command format**

  :SBUS:RS232:BITorder {LSBFirst | MSBFirst}

  :SBUS:RS232:BITorder?

- **Functional description**

  Set RS232 bus decoding bit order of the oscilloscope, which includes LSBFirst and MSBFirst.

- **Return format**

  Query returns {LSBFirst | MSBFirst}.

- **For example**

  :SBUS:RS232:BITorder LSBF Set RS232 bus decoding bit order to LSBF

  :SBUS:RS232:BITorder? Query returns LSBFirst

:SBUS:RS232:SOURce

- **Command format**

  :SBUS:RS232:SOURce { CHANnel1 | CHANnel2 }

  :SBUS:RS232:SOURce?

- **Functional description**

  Set RS232 bus decoding source of the oscilloscope.

- **Return format**

Query returns { CHANnel1 | CHANnel2 }.

- **For example**

:SBUS:RS232:SOURce CHANnel1 Set CH1 as RS232 bus decoding source

:SBUS:RS232:SOURce? Query returns CHANnel1

:SBUS:RS232:POLarity

- **Command format**

:SBUS:RS232:POLarity { POSitive | NEGative }

:SBUS:RS232:POLarity?

- **Functional description**

Set RS232 bus decoding polarity of the oscilloscope, including POSitive (rising) and NEGative (falling).

- **Return format**

Query returns { POSitive | NEGative }.

- **For example**

:SBUS:RS232:POLarity POSitive

Set RS232 bus decoding polarity to POSitive

:SBUS:RS232:POLarity? Query returns POSitive

:SBUS:RS232:PARity

- **Command format**

:SBUS:RS232:PARity {EVEN | ODD | NONE}

:SBUS:RS232:PARity?

- **Functional description**

Set RS232 bus parity of the oscilloscope.

- **Return format**

Query returns {EVEN | ODD | NONE}.

- **For example**

:SBUS:RS232:PARity ODD Set RS232 bus parity to ODD

:SBUS:RS232:PARity? Query returns ODD

:SBUS:RS232:DATA:BIT

- **Command format**

:SBUS:RS232:DATA:BIT {5 | 6 | 7 | 8 }

:SBUS:RS232:DATA:BIT?

- **Functional description**

  Set RS232 bus data bit of the oscilloscope.

- **Return format**

  Query returns {5 | 6 | 7 | 8 }.

- **For example**

:SBUS:RS232:DATA:BIT 6	Set RS232 data bit to 6
:SBUS:RS232:DATA:BIT?	Query returns 6

:SBUS:RS232:STOP:BIT

- **Command format**

  :SBUS:RS232:STOP:BIT {1 | 2 }

  :SBUS:RS232:STOP:BIT?

- **Functional description**

  Set RS232 bus stop bit of the oscilloscope.

- **Return format**

  Query returns {1 | 2 }.

- **For example**

:SBUS:RS232:STOP:BIT 6	Set RS232 stop bit to 6
:SBUS:RS232:STOP:BIT?	Query returns 6

:SBUS:RS232:DATA

- **Command format**

  :SBUS:RS232:DATA <value>

  :SBUS:RS232:DATA?

- **Functional description**

  Set RS232 bus trigger data of the oscilloscope. The parameter is 0 or 1 which represent the binary character data, its range is related to the value set by the command :SBUS:RS232:DATA:BIT.

- **Return format**

  Query returns the binary character data.

- **For example**

:SBUS:RS232:DATA "1111111"	Set RS232 bus data to 0x7F
:SBUS:RS232:DATA?	Query returns 1111111

:SBUS:RS232:QUALifier

- **Command format**

  :SBUS:RS232:QUALifier {BEGFrame | ERRFrame | ECCError | DATA}

  :SBUS:RS232:QUALifier?

- **Functional description**

  Set RS232 bus trigger condition of the oscilloscope.

- **Return format**

  Query returns {BEGFrame|ERRFrame|ECCError|DATA}.

- **For example**

:SBUS:RS232:QUALifier ERRF	Set RS232 bus condition to ERRF
:SBUS:RS232:QUALifier?	Query returns ERRFrame

I^2C

:SBUS:I2C:CLOCk:SOURce

- **Command format**

  :SBUS:I2C:CLOCk:SOURce { CHANnel1 | CHANnel2 }

  :SBUS:I2C:CLOCk:SOURce?

- **Functional description**

  Set I^2C bus clock source of the oscilloscope.

- **Return format**

  Query returns { CHANnel1 | CHANnel2 }。

- **For example**

:SBUS:I2C:CLOCk:SOURce CHANnel1	Set CH1 as I^2C bus clock source
:SBUS:I2C:CLOCk:SOURce?	Query returns CHANnel1

:SBUS:I2C:DATA:SOURce

- **Command format**

  :SBUS:I2C:DATA:SOURce { CHANnel1 | CHANnel2 }

  :SBUS:I2C:DATA:SOURce?

- **Functional description**

  Set I^2C bus data source of the oscilloscope.

- **Return format**

  Query returns { CHANnel1 | CHANnel2 }.

- **For example**

:SBUS:I2C:DATA:SOURce CHANnel1	Set CH1 as I²C data clock source
:SBUS:I2C:DATA:SOURce?	Query returns CHANnel1

:SBUS:I2C:ASIZe

- **Command format**

  :SBUS:I2C:ASIZe {7 | 10}

  :SBUS:I2C:ASIZe?

- **Functional description**

  Set I²C bus address bit wide of the oscilloscope.

- **Return format**

  Query returns {7 | 10}.

- **For example**

:SBUS:I2C:ASIZe 7	Set I²C bus address bit wide to 7
:SBUS:I2C:ASIZe?	Query returns 7

:SBUS:I2C:ADIRection

- **Command format**

  :SBUS:I2C:ADIRection { READ | WRITE }

  :SBUS:I2C:ADIRection?

- **Functional description**

  set I²C bus address direction of the oscilloscope.

- **Return format**

  Query returns { READ | WRITE }.

- **For example**

:SBUS:I2C:ADIRection READ	Set I²C bus address direction to READ
:SBUS:I2C:ADIRection?	Query returns READ

:SBUS:I2C:ADDRess

- **Command format**

  :SBUS:I2C:ADDRess <value>

  :SBUS:I2C:ADDRess?

- **Functional description**

  Set I²C bus address of the oscilloscope. The parameter is 0 or 1 which represent the binary character data, its range is related to the value set by the command :SBUS:I2C:ASIZe, it is [0~2^addressbit − 1].

- **Return format**

  Query returns the address with binary character string

- **For example**

  :SBUS:I2C:ADDRess "1111111" Set I²C bus address to 0x7F

  :SBUS:I2C:ADDRess? Query returns 1111111

:SBUS:I2C:QUALifier

- **Command format**

  :SBUS:I2C:QUALifier {STARt | RESTart | STOP | LOSS | ADDRess | DATA | ADATA}

  :SBUS:I2C:QUALifier?

- **Functional description**

  Set I²C bus trigger condition of the oscilloscope.

- **Return format**

  Query returns {STARt | RESTart | STOP | LOSS | ADDRess | DATA | ADATA}.

- **For example**

  :SBUS:I2C:QUALifier STOP Set I²C bus trigger condition to STOP

  :SBUS:I2C:QUALifier? Query returns STOP

:SBUS:I2C:DATA:LEN

- **Command format**

  :SBUS:I2C:DATA:LEN <length>

  :SBUS:I2C:DATA:LEN?

- **Functional description**

  Set I²C bus trigger data length of the oscilloscope. It can take value from 1~5.

- **Return format**

  Query returns I²C bus trigger data length of the oscilloscope, it is integer data.

- **For example**

  :SBUS:I2C:DATA:LEN 2 Set I²C bus trigger data length to 2 bytes

  :SBUS:I2C:DATA:LEN? Query returns 2

:SBUS:I2C:DATA

- **Command format**

  :SBUS:I2C:DATA <value>

  :SBUS:I2C:DATA?

- **Functional description**

Set I2C bus data, the parameter is 0 or 1 which represent the binary character data, the data range is 0x0~0xFFFFFFFFFFFFFFFF.

- **Return format**

Query returns the data with binary character string.

- **For example**

:SBUS:I2C:DATA "1111111111111111" Set I^2C bus data to 0xFFFFF

:SBUS:I2C:DATA? Query returns 1111111111111111

SPI

:SBUS:SPI:CLOCk:SOURce

- **Command format**

:SBUS:SPI:CLOCk:SOURce { CHANnel1 | CHANnel2 }

:SBUS:SPI:CLOCk:SOURce?

- **Functional description**

Set SPI bus clock source of the oscilloscope.

- **Return format**

Query returns { CHANnel1 | CHANnel2 }.

- **For example**

:SBUS:SPI:CLOCk:SOURce CHANnel1

Set CH1 as SPI bus clock source

:SBUS:SPI:CLOCk:SOURce? Query returns CHANnel1

:SBUS:SPI:MOSI:SOURce

- **Command format**

:SBUS:SPI:MOSI:SOURce { CHANnel1 | CHANnel2 }

:SBUS:SPI:MOSI:SOURce?

- **Functional description**

Set SPI bus source is output from master, input from slaver

- **Return format**

Query returns { CHANnel1 | CHANnel2 }.

- **For example**

:SBUS:SPI:MOSI:SOURce CHANnel1

Set CH1 as the source of SPI output from master, input from slaver

:SBUS:SPI:MOSI:SOURce? Query returns CHANnel1

:SBUS:SPI:BITorder

- **Command format**

  :SBUS:SPI:BITorder {LSBFirst | MSBFirst}

  :SBUS:SPI:BITorder?

- **Functional description**

  Set SPI bus decoding byte order of the oscilloscope, including LSBFirst and MSBFirst.

- **Return format**

  Query returns {LSBFirst | MSBFirst}.

- **For example**

  :SBUS:SPI:BITorder LSBF Set SPI bus decoding byte order to LSB

  :SBUS:SPI:BITorder? Query returns LSBFirst

:SBUS:SPI:CLOCk:POLarity

- **Command format**

  :SBUS:SPI:CLOCk:POLarity {NEGative | POSitive}

  :SBUS:SPI:CLOCk:POLarity?

- **Functional description**

  Set SPI bus clock polarity of the oscilloscope, including POSitive (rising), NEGative (falling).

- **Return format**

  Query returns {NEGative | POSitive}.

- **For example**

  :SBUS:SPI:CLOCk:POLarity POSitive

  Set SPI bus clock polarity to POSitive

  :SBUS:SPI:CLOCk:POLarity? Query returns POSitive

:SBUS:SPI:MOSI:POLarity

- **Command format**

  :SBUS:SPI:MOSI:POLarity {NEGative | POSitive}

  :SBUS:SPI:MOSI:POLarity?

- **Functional description**

  Set the polarity of SPI MOSI (output from master, input from slave), including POSitive (rising) and NEGative (falling).

- **Return format**

  Query returns {NEGative | POSitive}.

- **For example**

  :SBUS:SPI:MOSI:POLarity POSitive Set the polarity of SPI MOSI to POSitive

  :SBUS:SPI:MOSI:POLarity? Query returns POSitive

:SBUS:SPI:WIDTh

- **Command format**

  :SBUS:SPI:WIDTh <width>

  :SBUS:SPI:WIDTh?

- **Functional description**

  Set SPI bus data bit width of the oscilloscope.

  <width> is integer data, the range is 4~32.

- **Return format**

  Query returns SPI bus data bit width.

- **For example**

  :SBUS:SPI:WIDTh 4 Set SPI bus data bit width to 4

  :SBUS:SPI:WIDTh? Query returns 4

:SBUS:SPI:FRAMelen

- **Command format**

  :SBUS:SPI:FRAMelen <len>

  :SBUS:SPI:FRAMelen?

- **Functional description**

  Set SPI bus data frame length of the oscilloscope.

  <len> is integer data, the range is 1~32，data width*data frame length cannot exceed 128 bits data.

- **Return format**

  Query returns SPI bus data frame length.

- **For example**

  :SBUS:SPI:FRAMelen 1 Set SPI bus data frame length to 1

  :SBUS:SPI:FRAMelen? Query returns 1

:SBUS:SPI:DATA

- **Command format**

  :SBUS:SPI:DATA <value>

  :SBUS:SPI:DATA?

- **Functional description**

  Set SPI bus data of the oscilloscope, the parameter is 0 or 1 which represent the binary character data, X represents unsure. The data range is 0x0~0Xffffffffffffffff.

- **Return format**

  Query returns the data with binary character string.

- **For example**

  :SBUS:SPI:DATA "X00X00X1" Set SPI bus data to X00X00X1

  :SBUS:SPI:DATA? Query returns X00X00X1

:SBUS:SPI:QUALifier

- **Command format**

  :SBUS:SPI:QUALifier {IDLE|IDLEDATA}

  :SBUS:SPI:QUALifier?

- **Functional description**

  Set SPI bus condition of the oscilloscope.

- **Return format**

  Query returns {IDLE|IDLEDATA}.

- **For example**

  :SBUS:SPI:QUALifier IDLE Set SPI bus condition to IDLE

  :SBUS:SPI:QUALifier? Query returns IDLE

:SBUS:SPI:TRIGger:TIMeout

- **Command format**

  :SBUS:SPI:TRIGger:TIMeout <vlaue>

  :SBUS:SPI:TRIGger:TIMeout?

- **Functional description**

  Set SPI bus trigger timeout of the oscilloscope, the parameter is integer. <vlaue> is equal to n * 4ns and value not exceed range [100ns,1s) . n takes value from [25,25*10^8].

- **Return format**

  Query returns trigger timeout value by scientific notation, the unit is s.

- **For example**

  :SBUS:SPI:TRIGger:TIMeout 100ns

  Set SPI bus trigger timeout value to 100 ns

  :SBUS:SPI:TRIGger:TIMeout? Query returns 1.000e-007

Explanation of Programming

This chapter is to describe troubleshooting in process of programming. If you meet any of the following problems, please handle them according to the related instructions.

Programming Preparation

Programming preparation is only applicable for using Visual Studio and LabVIEW development tools to programming under Windows operating system.

Firstly, user need to confirm that whether NI –VISA library is installed (it can be download from the website

https://www.ni.com/en-ca/support/downloads/drivers/download.ni-visa.html).

In this manual, the default installment path is C:\Program Files\IVI Foundation\VISA.

Build communication with PC via USB or LAN interface of the instrument, use USB data line to connect USB DEVICE port on the rear panel of the instrument with USB port of PC, or use LAN data line to connect LAN port on the rear panel of the instrument with LAN port of PC.

VISA Programming Example

There are some example in this section. Through these examples, user can know how to use VISA, and it can combined with the command of programming manual to realize the control of the instrument. With these examples, user can develop more applications.

VC++Example

- Environment: Window system, Visual Studio
- Description: Access the instrument via USBTMC and TCP/IP, and send "*IDN?" command on NI-VISA to query the device information.
- Steps:

1. Open Visual Studio software to create a new VC++ win32 console project.
2. Set project environment that can adjust NI-VISA library, which are static library and dynamic library.
3. Set the project environment for calling NI-VISA library, which are static library and dynamic library respectively.

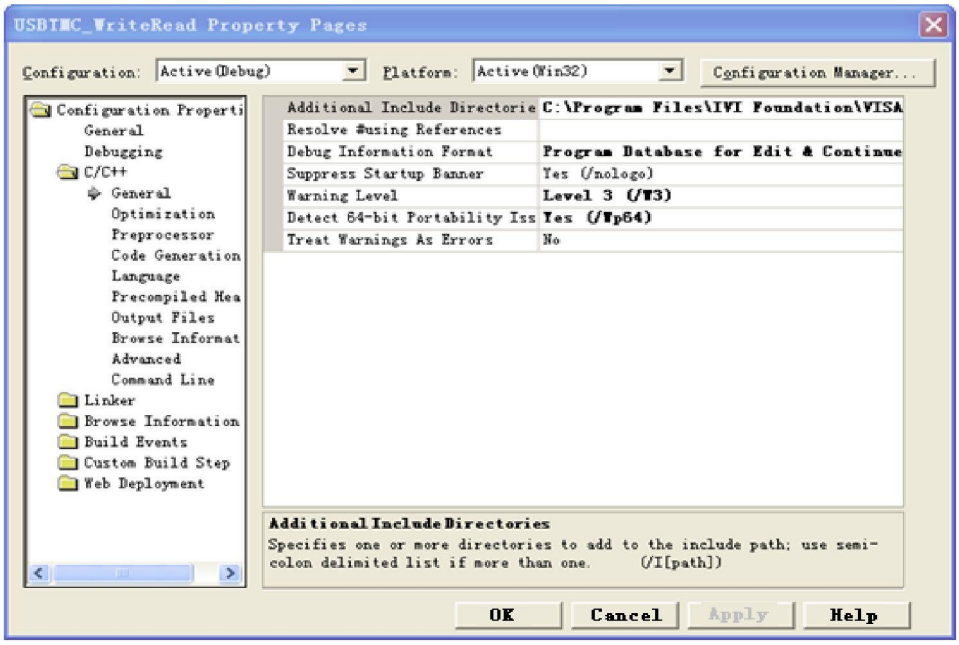
a) Static library

In NI-VISA installment path to find file visa.h, visatype.h and visa32.lib and copy them to the root path of VC++ project and add it to the project. Add two lines of code into file projectname.cpp as follows.

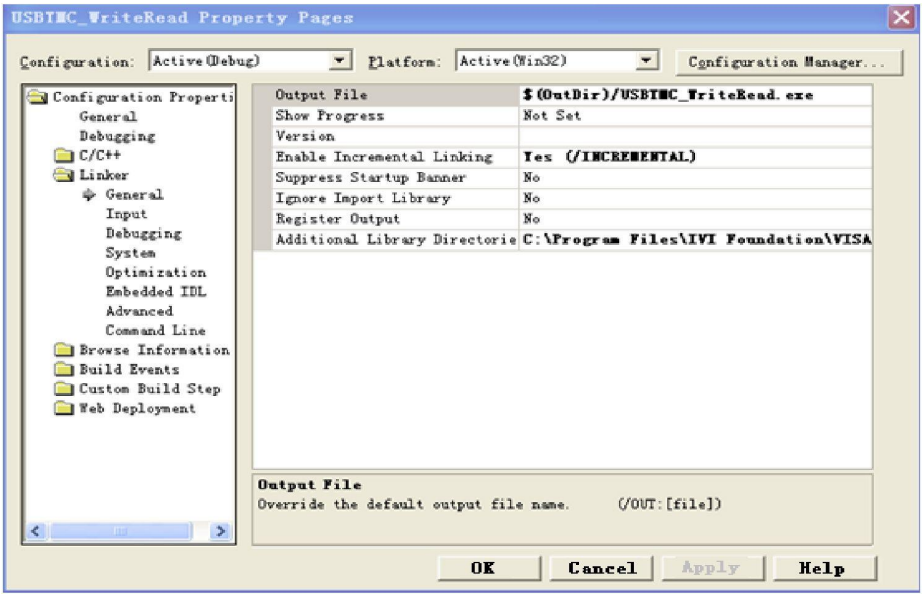
```
#include "visa.h"
#pragma comment(lib,"visa32.lib")
```

b) Dynamic library

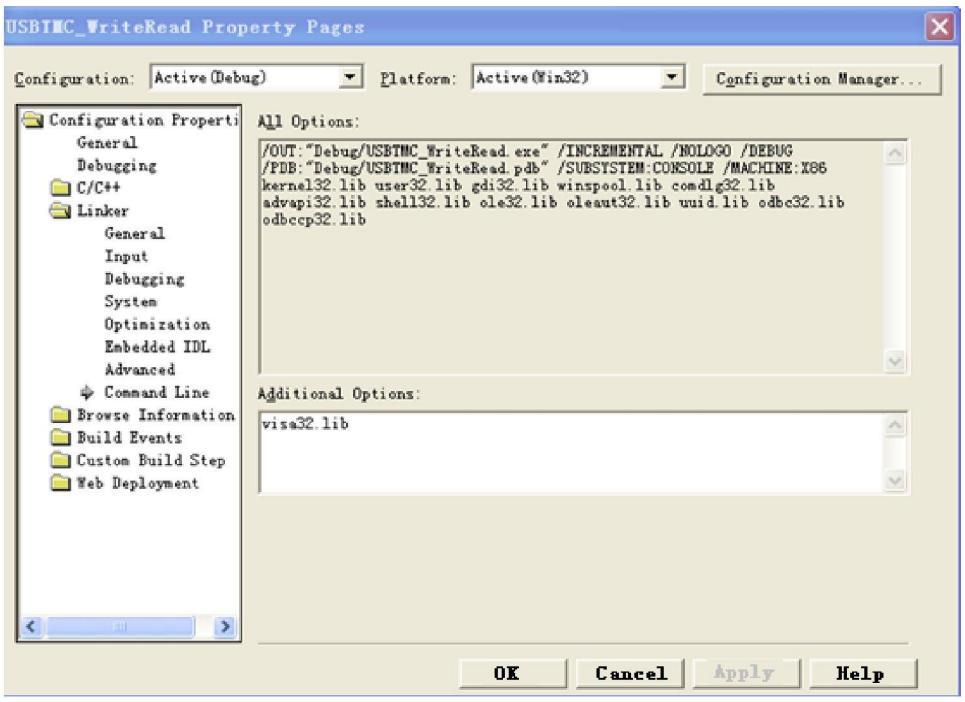
Press "project>>properties", select"c/c++---General" in attribute dialog on the left side, set the value of "Additional Include Directories" as the installment path of NI-VIS (such as C:\ProgramFiles\IVI Foundation\VISA\WinNT\include), as shown in the following figure.



Select "Linker-General" in attribute dialog on the lefts side, set the value of "Additional Library Directories" as the installment path of NI-VIS (such as C:\Program Files\IVI Foundation\VISA\WinNT\include), as shown in the following figure.



Select “Linker-Command Line “in attribute dialog on the left side, set the value of "Additional" as visa32.lib, as shown in the following figure.



Add file visa.h in projectname.cpp file

#include <visa.h>

1. Source code

a) USBTMC Example

```
int usbtmc_test()
{ /** This code demonstrates sending synchronous read & write commands
    * to an USB Test & Measurement Class (USBTMC) instrument using NI-VISA
    * The example writes the "*IDN?\n" string to all the USBTMC
    * devices connected to the system and attempts to read back
    * results using the write and read functions.
    * Open Resource Manager
    * Open VISA Session to an Instrument
    * Write the Identification Query Using viPrintf
    * Try to Read a Response With viScanf
    * Close the VISA Session*/
    ViSession defaultRM;
    ViSession instr;
    ViUInt32 numInstrs;
    ViFindList findList;
    ViStatus status;
    char instrResourceString[VI_FIND_BUFLEN];
    unsigned char buffer[100];
    int i;
    status = viOpenDefaultRM(&defaultRM);
    if (status < VI_SUCCESS)
    {
        printf("Could not open a session to the VISA Resource Manager!\n");
```

```
        return status;
}
/*Find all the USB TMC VISA resources in our system and store the number of resources in the system in
numInstrs.*/
status = viFindRsrc(defaultRM, "USB?*INSTR", &findList, &numInstrs, instrResourceString);
if (status<VI_SUCCESS)
{
        printf("An error occurred while finding resources. \nPress Enter to continue.");
        fflush(stdin);
        getchar();
        viClose(defaultRM);
        return status;
}
/** Now we will open VISA sessions to all USB TMC instruments.
*       We must use the handle from viOpenDefaultRM and we must
*       also use a string that indicates which instrument to open. This
*       is called the instrument descriptor. The format for this string
*       can be found in the function panel by right clicking on the
*       descriptor parameter. After opening a session to the
*       device, we will get a handle to the instrument which we
*       will use in later VISA functions. The AccessMode and Timeout
*       parameters in this function are reserved for future
*       functionality. These two parameters are given the value VI_NULL. */
for (i = 0; i < int(numInstrs); i++)
{
        if (i > 0)
        {
                viFindNext(findList, instrResourceString);
        }
        status = viOpen(defaultRM, instrResourceString, VI_NULL, VI_NULL, &instr);
        if (status < VI_SUCCESS)
        {
                printf("Cannot open a session to the device %d. \n", i + 1);
                continue;
        }
        /** At this point we now have a session open to the USB TMC instrument.
        *We will now use the viPrintf function to send the device the string "*IDN?\n",
        *asking for the device's identification. */
        char * cmmand = "*IDN?\n";
        status = viPrintf(instr, cmmand);
        if (status < VI_SUCCESS)
        {
                printf("Error writing to the device %d. \n", i + 1);
                status = viClose(instr);
                continue;
        }
```

```
            /** Now we will attempt to read back a response from the device to
            *the identification query that was sent. We will use the viScanf
            *function to acquire the data.
            *After the data has been read the response is displayed. */
            status = viScanf(instr, "%t", buffer);
            if (status < VI_SUCCESS)
            {
                printf("Error reading a response from the device %d. \n", i + 1);
            }
            else
            {
                printf("\nDevice %d: %s\n", i + 1, buffer);
            }
            status = viClose(instr);
        }
        /*Now we will close the session to the instrument using viClose. This operation frees all
        system resources.*/
        status = viClose(defaultRM);
        printf("Press Enter to exit.");
        fflush(stdin);
        getchar();
        return 0;
    }

    int _tmain(int argc, _TCHAR* argv[])
    {
        usbtmc_test();
        return 0;
    }
```

b) TCP/IP Example

```
    int tcp_ip_test(char *pIP)
    {
        char outputBuffer[VI_FIND_BUFLEN];
        ViSession defaultRM, instr;
        ViStatus status;
        /* First we will need to open the default resource manager. */
        status = viOpenDefaultRM(&defaultRM);
        if (status < VI_SUCCESS)
        {
            printf("Could not open a session to the VISA Resource Manager!\n");
        }
        /* Now we will open a session via TCP/IP device */
        char head[256] = "TCPIP0::";
        char tail[] = "::inst0::INSTR";
        strcat(head, pIP);
```

```
        strcat(head, tail);
        status = viOpen(defaultRM, head, VI_LOAD_CONFIG, VI_NULL, &instr);
        if (status < VI_SUCCESS)
        {
            printf("An error occurred opening the session\n");
            viClose(defaultRM);
        }
        status = viPrintf(instr, "*idn?\n");
        status = viScanf(instr, "%t", outputBuffer);
        if (status < VI_SUCCESS)
        {
            printf("viRead failed with error code: %x \n", status);
            viClose(defaultRM);
        }
        else
        {
            printf("\nMesseage read from device: %*s\n", 0, outputBuffer);
        }
        status = viClose(instr);
        status = viClose(defaultRM);
        printf("Press Enter to exit.");
        fflush(stdin);
        getchar();
        return 0;
    }
    int _tmain(int argc, _TCHAR* argv[])
    {
        printf("Please input IP address:");
        char ip[256];
        fflush(stdin);
        gets(ip);
        tcp_ip_test(ip);
        return 0;
    }
```

C# Example

- Environment: Window system, Visual Studio
- Description: Access the instrument via USBTMC and TCP/IP, and send "*IDN?" command on NI-VISA to query the device information.
- Steps:

1. Open Visual Studio software and create a new C# console project.
2. Add C# quote Ivi.Visa.dll and NationalInstruments.Visa.dll of VISA.

3. Source code

a) USBTMC Example

```
class Program
{
    void usbtmc_test()
    {
        using (var rmSession = new ResourceManager())
        {
            var resources = rmSession.Find("USB?*INSTR");
            foreach (string s in resources)
            {
                try
                {
                    var mbSession = (MessageBasedSession)rmSession.Open(s);
                    mbSession.RawIO.Write("*IDN?\n");
                    System.Console.WriteLine(mbSession.RawIO.ReadString());
                }
                catch (Exception ex)
                {
                    System.Console.WriteLine(ex.Message);
                }
            }
        }
    }

    void Main(string[] args)
    {
        usbtmc_test();
    }
}
```

b) TCP/IP Example

```
class Program
 {
    void tcp_ip_test(string ip)
    {
        using (var rmSession = new ResourceManager())
        {
            try
            {
                var resource = string.Format("TCPIP0::{0}::inst0::INSTR", ip);
                var mbSession = (MessageBasedSession)rmSession.Open(resource);
                mbSession.RawIO.Write("*IDN?\n");
                System.Console.WriteLine(mbSession.RawIO.ReadString());
```

```
                }
                catch (Exception ex)
                {
                    System.Console.WriteLine(ex.Message);
                }
            }
        }

        void Main(string[] args)
        {
            tcp_ip_test("192.168.20.11");
        }
    }
```

VB Example

- Environment: Window system, Microsoft Visual Basic 6.0.
- Description: Access the instrument via USBTMC and TCP/IP, and send "*IDN?" command on NI-VISA to query the device information.
- Steps:

1. Open Visual Basic software and create a new standard application program project.
2. Set the project environment that can adjust NI-VISA library, press Existing tab of Project>>Add Existing Item, in file "include" of NI-VISA installment path to find file visa32.bas and add this file, as shown in the following

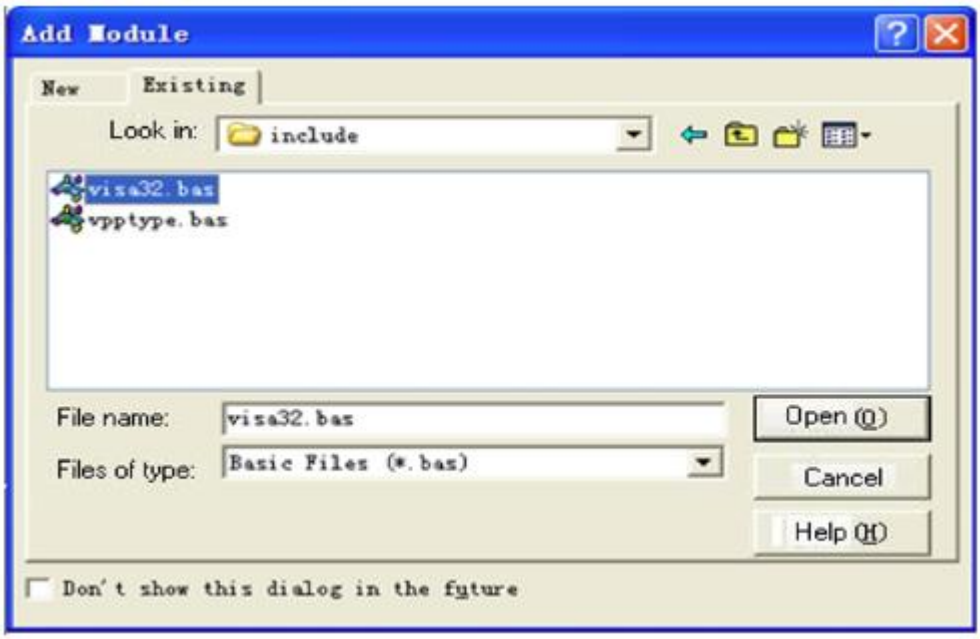


figure.

3. Source code

a) USBTMC Example

```
PrivateFunction usbtmc_test() AsLong
' This code demonstrates sending synchronous read & write commands
' to an USB Test & Measurement Class (USBTMC) instrument using NI-VISA
' The example writes the "*IDN?\n" string to all the USBTMC
' devices connected to the system and attempts to read back
' results using the write and read functions.
' The general flow of the code is
' Open Resource Manager
' Open VISA Session to an Instrument
' Write the Identification Query Using viWrite
' Try to Read a Response With viRead
' Close the VISA Session

Const MAX_CNT = 200
Dim defaultRM AsLong
Dim instrsesn AsLong
Dim numInstrs AsLong
Dim findList AsLong
Dim retCount AsLong
Dim status AsLong
Dim instrResourceString AsString *VI_FIND_BUFLEN
Dim Buffer AsString * MAX_CNT
Dim i AsInteger

' First we must call viOpenDefaultRM to get the manager
' handle. We will store this handle in defaultRM.
status = viOpenDefaultRM(defaultRM)
If(status < VI_SUCCESS) Then
    resultTxt.Text = "Could not open a session to the VISA Resource Manager!"
    usbtmc_test = status
ExitFunction
EndIf

' Find all the USB TMC VISA resources in our system and store the
' number of resources in the system in numInstrs.
status = viFindRsrc(defaultRM, "USB?*INSTR", findList, numInstrs, instrResourceString)
If (status < VI_SUCCESS) Then
    resultTxt.Text = "An error occurred while finding resources."
    viClose(defaultRM)
    usbtmc_test = status
```

```
ExitFunction
EndIf

' Now we will open VISA sessions to all USB TMC instruments.
' We must use the handle from viOpenDefaultRM and we must
' also use a string that indicates which instrument to open.  This
' is called the instrument descriptor.  The format for this string
' can be found in the function panel by right clicking on the
' descriptor parameter. After opening a session to the
' device, we will get a handle to the instrument which we
' will use in later VISA functions.  The AccessMode and Timeout
' parameters in this function are reserved for future
' functionality.  These two parameters are given the value VI_NULL.
For i = 0 To numInstrs
If (i > 0) Then
    status = viFindNext(findList, instrResourceString)
EndIf
    status = viOpen(defaultRM, instrResourceString, VI_NULL, VI_NULL, instrsesn)
If (status < VI_SUCCESS) Then
    resultTxt.Text = "Cannot open a session to the device " + CStr(i + 1)
GoTo NextFind
EndIf

' At this point we now have a session open to the USB TMC instrument.
' We will now use the viWrite function to send the device the string "*IDN?",
' asking for the device's identification.
status = viWrite(instrsesn, "*IDN?", 5, retCount)
If (status < VI_SUCCESS) Then
    resultTxt.Text = "Error writing to the device."
    status = viClose(instrsesn)
GoTo NextFind
EndIf

' Now we will attempt to read back a response from the device to
' the identification query that was sent.  We will use the viRead
' function to acquire the data.
' After the data has been read the response is displayed.
status = viRead(instrsesn, Buffer, MAX_CNT, retCount)
If (status < VI_SUCCESS) Then
    resultTxt.Text = "Error reading a response from the device." + CStr(i + 1)
Else
    resultTxt.Text = "Read from device: " + CStr(i + 1) + " " + Buffer
EndIf
    status = viClose(instrsesn)
Next i
```

```
' Now we will close the session to the instrument using
' viClose. This operation frees all system resources.
status = viClose(defaultRM)
usbtmc_test = 0
EndFunction
```

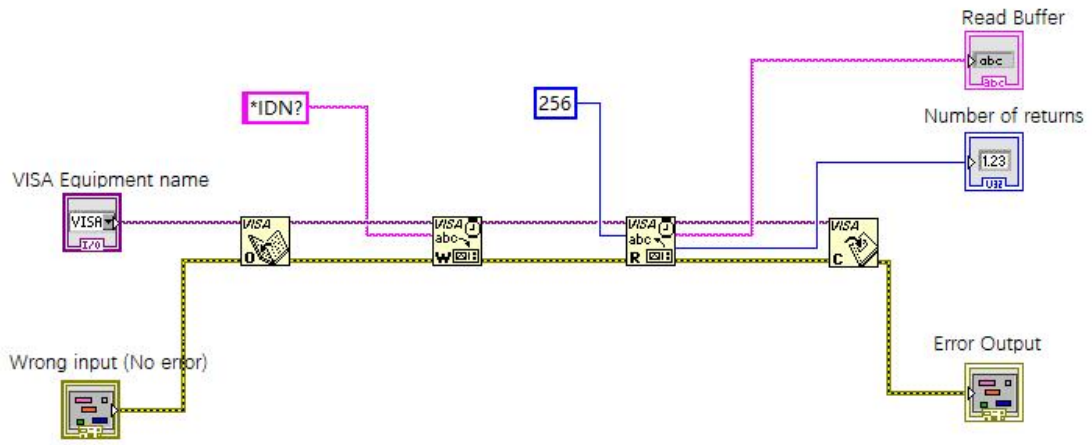
b) TCP/IP Example

```
PrivateFunction tcp_ip_test(ByVal ip AsString) AsLong
Dim outputBuffer AsString * VI_FIND_BUFLEN
Dim defaultRM AsLong
Dim instrsesn AsLong
Dim status AsLong
Dim count AsLong

' First we will need to open the default resource manager.
status = viOpenDefaultRM(defaultRM)
If (status < VI_SUCCESS) Then
    resultTxt.Text = "Could not open a session to the VISA Resource Manager!"
    tcp_ip_test = status
ExitFunction
EndIf

' Now we will open a session via TCP/IP device
status = viOpen(defaultRM, "TCPIP0::" + ip + "::inst0::INSTR", VI_LOAD_CONFIG, VI_NULL, instrsesn)
If (status < VI_SUCCESS) Then
    resultTxt.Text = "An error occurred opening the session"
    viClose(defaultRM)
    tcp_ip_test = status
ExitFunction
EndIf
status = viWrite(instrsesn, "*IDN?", 5, count)
If (status < VI_SUCCESS) Then
    resultTxt.Text = "Error writing to the device."
EndIf
    status = viRead(instrsesn, outputBuffer, VI_FIND_BUFLEN, count)
If (status < VI_SUCCESS) Then
    resultTxt.Text = "Error reading a response from the device." + CStr(i + 1)
Else
    resultTxt.Text = "read from device:" + outputBuffer
EndIf
    status = viClose(instrsesn)
    status = viClose(defaultRM)
    tcp_ip_test = 0
EndFunction
```

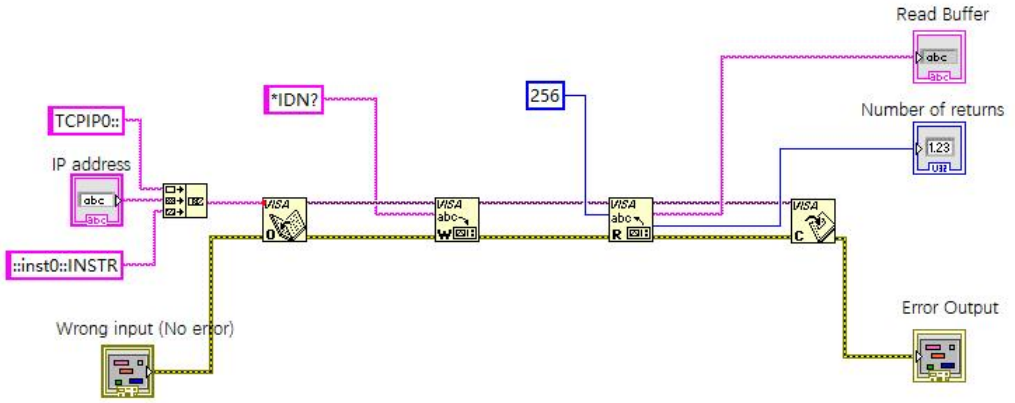
LabVIEW Example

- Environment: Window system, LabVIEW
- Description: Access the instrument via USBTMC and TCP/IP, and send "*IDN?" command on NI-VISA to query the device information.
- Steps:

1. Open LabVIEW software and create a VI file.
2. Add control, press the front panel interface, select and add VISA resource name, error input, error output and partial identifier on control flow diagram.
3. Open diagram, press VISA resource name and then select and add function VISA Write, VISA Read, VISA Open and VISA Close on pop-out menu.
4. VI open a VISA session of USBTMC device and wrote *IDN? command and read back the response value. When all communication is complete, VI will close the VISA session.



5. Communication with the device via TCP/IP is similar with USBTMC, it need to set VISA write and read function to synchronous I/O, set LabVIEW to asynchronous IO by default. Right click on the node and select "Synchronous I/O Mode>>Synchronous" from shortcut menu to enable synchronous writing or reading of data, as shown in the following figure.



MATLAB Example

- Environment: Window system, MATLAB
- Description: Access the instrument via USBTMC and TCP/IP, and send "*IDN?" command on NI-VISA to query the device information.
- Steps:

1. Open MATLAB software, click File>>New>>Script on Matlab interface to create an empty M file.
2. Source code:

a) USBTMC Example

```
function usbtmc_test()
% This code demonstrates sending synchronous read & write commands
% to an USB Test & Measurement Class (USBTMC) instrument using
% NI-VISA

%Create a VISA-USB object connected to a USB instrument
vu = visa('ni','USB0::0x5345::0x1234::SN20220718::INSTR');

%Open the VISA object created
fopen(vu);

%Send the string "*IDN?",asking for the device's identification.
fprintf(vu,'*IDN?');

%Request the data
```

```
outputbuffer = fscanf(vu);
disp(outputbuffer);

%Close the VISA object
fclose(vu);
delete(vu);
clear vu;

end
```

b) TCP/IP Example

```
function tcp_ip_test()
% This code demonstrates sending synchronous read & write commands
% to an TCP/IP instrument using NI-VISA
%Create a VISA-TCPIP object connected to an instrument

%configured with IP address.
vt = visa('ni',['TCPIP0::','192.168.20.11','::inst0::INSTR']);

%Open the VISA object created

fopen(vt);

%Send the string "*IDN?",asking for the device's identification.
fprintf(vt,'*IDN?');

%Request the data
outputbuffer = fscanf(vt);
disp(outputbuffer);

%Close the VISA object
fclose(vt);
delete(vt);
clear vt;

end
```

Python Example

- Environment: Window system, Python3.8, PyVISA 1.11.0
- Description: Access the instrument via USBTMC and TCP/IP, send "*IDN?" command on NI-VISA

to query the device information.

- Steps:

1. Install python firs, and then turn on Python script compiling software, create an empty test.py file.
2. Use pip install PyVISA instruction to install PyVISA, if it cannot install, please refer to this link (https://pyvisa.readthedocs.io/en/latest/)
3. Source code:

a) USBTMC Example

```
import pyvisa
rm = pyvisa.ResourceManager()
rm.list_resources()
my_instrument = rm.open_resource('USB0::0x5345::0x1234::SN20220718::INSTR')
print(my_instrument.query('*IDN?'))
```

b) TCP/TP Example

```
import pyvisa
rm = pyvisa.ResourceManager()
rm.list_resources()
my_instrument = rm.open_resource('TCPIP0::192.168.20.11::inst0::INSTR')
print(my_instrument.query('*IDN?'))
```

Programming Application Exam

Set bandwidth limit

When observing the low-frequency signal, it is necessary to reduce the high-frequency noise in the signal, then attenuate the high-frequency signal above 20 MHz in the signal. It can use the following command to set bandwidth limit, such as set channel 1,

CHANnel1:BWLimit ON #Turn on bandwidth limit of channel 1.

CHANnel1:BWLimit?

#Query returns 1, it represents bandwidth limit of channel 1 has turned on.

Set offset voltage

Set the channel's offset voltage, it can use the following command to setup, such as offset voltage of channel 1,

CHANnel1:OFFSet 1V #CH1 move up 1 V, then offset voltage is 1 V.

CHANnel1:OFFSet? #Query the channel's offset voltage.

Set volts/div scale

Set volts/div scale of channel, it can be set as the following command, such as set volts/div scale of channel 1,

CHANnel1:SCALe 500 mV #Set volts/div scale of channel 1 to 500 mV.

CHANnel1:SCALe? #Query volts/div scale value of channel 1.

Set timebase scale

Set timebase scale of the oscilloscope, it can be set as the following command,

TIMebase:SCALe 0.005 #Set timebase scale of the oscilloscope to 5 ms.

TIMebase:SCALe? #Query timebase scale of the oscilloscope.

Query amplitude value

User can run the following command to query the amplitude measurement results without opening the measurement window, such as query amplitude value of channel 1 waveform,

MEASure:VPP? CHANnel1 #Query amplitude value of CH1 waveform.

Query rising delay time value

User can run the following command to query rising delay measuring time without opening the measurement window, such as query rising delay value of channel 1 and channel 2,

MEASure:PDELay? CHANnel1, CHANnel2

#Query rising delay value of CH1 and CH2.

Appendix 1: Keypad List

Key	Functional Description	LED
CH1	Channel 1 switch	√
CH2	Channel 2 switch	√
MATH	Mathematical operation and menu	√
AUTO	The control values of the oscilloscope are automatically set to condign display the waveform for observation	
RS	Control the oscilloscope's running status, continuous send this command, the oscilloscope can switch to stop or run	√
SINGle	Single trigger	√
TMENu	Trigger menu	
HMENu	Horizontal system menu	
MENu	Menu display switch	
F1	Select the first menu item of the current menu	
F2	Select the second menu item of the current menu	
F3	Select the third menu item of the current menu	
F4	Select the fourth menu item of the current menu	
F5	Select the fifth menu item of the current menu	
F6	Select the sixth menu item of the current menu	
PGDN	Next page	
MEASure	Measurement function	
CURSor	Cursor measurement function and menu	√
ACQuire	Sampling menu	
DISPlay	Display menu	
STORage	Storage menu	
UTILity	System auxiliary menu	
HOMe	Home menu	
BUS	Bus menu	√
HELP	Help menu	
NAVigate	Navigation menu	
DEFault	Reset setting	
PSCReen	One-key print or one-key save the screen image	
REF	Reference waveform menu	√
LEFT	Left-function key	√
STOP	Stop key	√
RIGHt	Right-function key	√

TFORe	Force trigger key	
MODE	Mode switching key	
FKNob	Multi-function rotary knob	
FKNLeft	Multi-function left rotary knob	
FKNRight	Multi-function right rotary knob	
VPKNob	Vertical rotary knob	
VPKNLeft	Vertical left rotary knob	
VPKNRight	Vertical right rotary knob	
HPKNob	Horizontal rotary knob	
HPKNLeft	Horizontal left rotary knob	
HPKNRight	Horizontal right rotary knob	
TPKNob	Trigger rotary knob	
TPKNLeft	Trigger left rotary knob	
TPKNRight	Trigger right rotary knob	
VBKNob	Voltage reference rotary knob	
VBKNLeft	Voltage reference left rotary knob	
VBKNRight	Voltage reference right rotary knob	
TBKNob	Timebase reference rotary knob	
TBKNLeft	Timebase reference left rotary knob	
TBKNRight	Timebase reference right rotary knob	
ASTAtus	Automatic indicator, no key, only has light	√
NSTAtus	Normal indicator, no key, only has light	√
SSTAtus	Single indicator, no key, only has light	√
FSTAtus	Multi-function rotary knob indicator, no key, only has light	√

Appendix 2: IEEE 488.2 Binary Data Format

DATA is data flow, other is ASCII character, as shown in the following figure <#812345678 + DATA + \n>

Start (1Byte)	Length Bit Wide (1Byte)	Total Data Length (Bit Width Byte)	DATA (n Byte)	End (1Byte)
#	x	x x x x x x x x	.....	\n